Interface

2003 April

Contents 2003 April

[表紙デザイン：㈱プランニング・ロケッツ]

特集

480Mbps対応USBターゲットからホストシステムの設計まで

Interface

Contents 2003 April

話題のテクノロジ解説

ショウレポート&コラム

一般解説&連載

■情報のページ

エレクトロニクスの総合展示会

インターネプコンワールド JAPAN 2003

北村俊之

エレクトロニクスの製造，実装に関する専門技術展である「インターネプコンワールド JAPAN 2003」が1月22日(水)～24日(金)の3日間，東京ビッグサイトで開催された．主催はリードエグジビジョンジャパン(株)．同展示会は，1972年にエレクトロニクス製造技術展としてスタートし，世界8か国11か所で開催されている．「第32回インターネプコン・ジャパン」，「第20回エレクトロテスト・ジャパン」を中心に，半導体パッケージ製造に必要な装置，部品を一堂に集めた「第4回半導体パッケージング技術展」，プリント配線板や電子部品を展示する「第4回プリント配線板EXPO」，「第4回電子コンポーネントEXPO」の5展示会に加え，光通信システム，デバイスを一堂に展示する「第3回ファイバーオプティクスEXPO」を併催し，出展社数も850社以上と今までにない大規模な開催となった．

〔写真1〕入場口のようす

最終的な来場者数は，3日間で52,593人となった(**写真1**)．今回は，「インターネプコン・ジャパン」および「エレクトロテスト・ジャパン」にスポットをあててレポートする．

● インターネプコン・ジャパン

〔写真2〕シュロニガージャパンのMS9600

シュロニガージャパンは，新製品である全自動カット&ストリップ装置「MS9600」を中心に，ケーブルから光ファイバの高精度加工装置までを幅広く展示していた(**写真2**)．日本ガーターは，チップ部品用高速テーピング機，オートリールチェンジャ，ピーリングテスタなどを多数展示しており，高速テーピング機は0.15秒/個の処理時間を実現しているという．パンドウイットコーポレーション日本支社は，ノンハロゲンダクトシリーズNNCタイプの発表を行っており，来場者の注目を集めていた．同シリーズは，燃焼時に塩素ガスを発生せず，環境にやさしい製品とのことである．

〔写真3〕ノードソンアシムテックの防湿絶縁剤塗布装置

ノードソンアシムテックは，防湿絶縁剤塗布装置に注目が集まっていた(**写真3**)．日本オートマチックマシンは，圧着タイプ業界最小の0.75mmピッチ超低背型SSJコネクタの展示および全自動端子圧着挿入機「JN01SS-IS-2C」による加工実演を行っていた．

〔写真4〕エム・シー・エムのブース

電線の多様化に対応するワイヤストリッパの新製品「R927」，「C1010」を中心に展示を行っていたのが，エム・シー・エムである．全機種で切り込み径のディジタル表示を可能にしているのが特徴であるという(**写真4**)．オリムベクスタは，電線のカット，ストリップ，ヒート，テストなどの多彩なニーズに応えるハーネス加工機やテスタ，ケーブルストリッパの展示を行っていた．

● エレクトロテスト・ジャパン

〔写真5〕レーザーテックのコンフォーカル顕微鏡HD100D

レーザーテックは，低倍率時での高い3次元分解能を実現したコンフォーカル顕微鏡「HD100D」の展示を行っていた．同製品では2光束干渉対物レンズを使用することで，視野がおよそ2mm時に10nm台の分解能を実現している(**写真5**)．

〔写真6〕DALSA社のエリアスキャンカメラ

ミツトヨは，新製品のニューコンセプト二次元画像測定機「QUICK IMAGEシリーズ」を中心に，非接触三次元CNC画像測定機，ユニバーサル顕微鏡などの展示を行っていた．伊藤忠テクノサイエンスは，DALSA社のエリアスキャンカメラ2機種を展示していた．「DALSTAR-SA 1M28」は最大10万フレームの部品読み出し機能とストップアクションにより，高速動体にも対応している(**写真6**)．

鷹山は，高速画像処理装置やワイヤレス画像伝送システムを中心に展示を行っており，新製品である「FX-III」は来場者の関心が高いという．オムロンは，基板はんだ検査装置，基板外観検査装置などを中心に展示を行っており，今回はじめてX線方式の検査装置の展示を行っていた．

〔写真7〕ナックイメージテクノロジーのハイスピードカメラMEMRECAMfx RX-3

ナックイメージテクノロジーでは，ハイスピードカメラ「MEMRECAMfx RX-3」を中心に展示を行っていた．同製品は生産ラインで発生するトラブルを高速撮影し，スローモーション再生することが可能なカメラシステムとのことである(**写真7**)．キヤノンシステムソリューションズ(旧住友金属システムソリュ一ションズ)は，画像処理検証ツール「OpenLogic」，「Image Processing Tools」など画像処理関連ツールやボードの展示を行っていた．

エスペックは，イオンマイグレーションや絶縁抵抗，動体抵抗などの特性に合わせた評価試験システムを展示していた．オカノ電機が展示していたのは，ディジタルオシロスコープ内蔵で，通信機能を搭載したオールラウンドテスタ，複雑な形状やデリケートな部品を高速で整列，搭載するチップ整列装置などである．

〔写真8〕ソキアファインシステムのSMIC-300

ソキアファインシステムでは，小型マスクや光コネクタ，マイクロマシン関係の小型精密部品などをレーザ干渉計による測長方式で非接触高精度自動測定する「SMIC-300」が来場者の注目を集めていた(**写真8**)．

Show&News Digest

Electronic Design and Solution Fair 2003

■日時：2003年1月30日（木）～31日（金）
■場所：パシフィコ横浜（神奈川県横浜市）

電子機器をLSI設計技術の展示会である，Electronic Design and Solution Fair 2003 with FPGA/PLD Design Conference（EDS Fair 2003）が開催された．

特別企画「スペシャルトークショー」では，女優の菊川怜が登場し，電子設計技術の進歩に興味を示すとともに，元気がないといわれる国内半導体業界に対してエールを送っていた．

キューウェーブ（http://www.que-wave.com/）は，ディジタル通信システム用高速RTL設計ツールQsim System，サイクルベース高速RTLシミュレータQsim Viewを展示していた．とくにQsim Viewは，イベントドリブンタイプのシミュレータより100倍以上高速だということだった．

アイピーフレックスのDAP/DNAリコンフィギュラブル・プロセッサは，アプリケーションに応じてハードウェアを1クロックで再構築できるプロセッサである．会場では，AM/FMチューナ，MP3プレーヤを1チップに収め，これらを切り替えて再生を行うデモを行っていた．また，新製品として0.13μmプロセスを用いたDAP/DNA-Tを2003年第3四半期に投入する予定とのことだ．

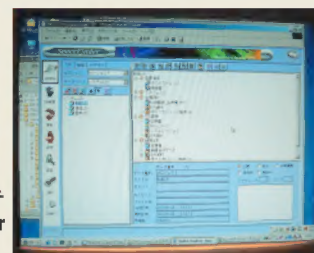
ダイキン工業システムのEDAファーム構築ソリューションProject∞（インフィニ）は，プロジェクト/部門ごとのリソース配分を最適化するソフトウェアである．複数のCPU資源/ストレージ資源などを管理し，最適な割り当てを行うことにより資源のムダをなくすことができる．また，EDAツールのライセンスを管理して，余剰ライセンスを活用することもできるというユニークなツールだ．ほかには，WebベースでR&Dデータの共有を実現するSpeedFinder Version3なども展示していた．

会場では，設計技術に関するプレゼンテーションを行うIPフリーマーケットが開催された．（有）ひまわりによる簡易WebサーバIPは，TCP/IPとHTTPを実現するIPである．FPGAなどにNE2000互換チップを接続するだけでWebサーバの機能を実現するというもの（http://www2g.biglobe.ne.jp/~himawari/kaisya/index.htm）である．HTMLデータは専用のスクリプトを用いてHDLに変換し，格納するとのことだ．

三菱電機システムLSI事業化推進センターによる32ビットRISCマイクロプロセッサM32Rソフトマクロは，M32RをIPで実現したものである．MMU搭載版/非搭載版があり，会場ではMMU搭載版のIPを二つ搭載しマルチプロセッサ構成にしたうえでLinuxを動作させるデモを行っていた．研究・教育目的用途にかぎりVDEC（http://www.vdec.u-tokyo.ac.jp/）にて無償公開している．

そのほかには広島市立大学大学院情報科学研究科によるSuperH命令セット互換プロセッサコアや，各社によるDES/AESの暗号化/復号化IP，CRC符号化IPなどの発表が行われた．

アイピーフレックスのDAP/DNAリコンフィギュラブル・プロセッサ

ダイキン工業システムのSpeedFinder Version3

トークショーに登場した菊川怜

キューウェーブのRTLシミュレータQsim View

Synopsysのブース

M32Rソフトマクロによるlinux．マルチプロセッサ構成のためペンギンが2匹表示されている

きっとエイエスピー，サーバベースドコンピューティングソフトウェア，GO-Globalを発売

■日時：2003年1月28日（火）
■場所：東京全日空ホテル（東京都港区）

きっとエイエスピー（http://www.kitasp.com/）は，GraphOn Corporation社が開発した，サーバベースドコンピューティングを可能にするソフトウェアGO-Globalを発売した．

GO-Globalはクライアントのマウス/キーボード操作情報をサーバへ転送し，サーバの画面更新情報をクライアントへ転送する．これにより，とくにWeb対応をうたっていない通常のアプリケーションでもリモート操作が可能になる．従来のソフトウェアでは画面のビットマップデータを転送していたため，インターネット越しの操作などでレスポンスが低下していたが，GO-Globalはサーバ上のAPIを仮想APIに変換し，それを独自プロトコルRapid-X（RXP）に乗せて転送するため，高速なレスポンスが確保できる．会場では，中国に設置したサーバ上でCADを動作させ，インターネット経由で操作を行い，軽快な動作をアピールしていた．

速度面以外にも，すでに存在する同様の製品よりも，動作するソフトウェアが多いとのことだ．動作可能なソフトウェアであるが，UNIXに関しては個別ソフトウェアについて動作保証を行うが，Windowsでは，その構造上，動作確認にとどまるとのことである．これらの動作保証/確認リストについては，今後Webなどで公開される．

代表取締役　松田利夫氏

ハッカーの常識的見聞録 ㉘

広畑由紀夫

今月の常識

Windows Media 9がやってくる！

☆ 本年1月29日より，ついに日本語版Windows Media 9がダウンロード開始になりました．
さて，Windows Media 9は，どのような進化をとげたのか，その点を考察します．

Windows Media 9でもっとも大きな改良点というと，5.1chへの対応があげられます．DVD-Videoに発した5.1chサウンド環境は，現在ではより進んだ6.1ch以上のチャネル数（スピーカ本数）を使用した音響へと進んでいますが，インターネットで配布される音声フォーマットは，特殊な例を除けば，ほとんどはステレオサウンドで記録・配信されていました．

また，5.1chのサウンドは，音響システムをもたない人にとってはなじみが少なく，しばらく前まではDVD-Videoを楽しむ一部の人達のシステムでしかありませんでした．それがここ数年間で価格が下落し，さらには5.1chスピーカシステムを標準セットで売り始めるメーカーが現れて一気に広まってきた感があります．

5.1chへの対応以外にも，従来のCDの音質劣化をさらに防ぎつつ圧縮を行う「Lossless」オーディオ圧縮を含め，低ビットレートの音質の向上から高ビットレートまでの多くのフォーマットが追加されました．

● 本格的なストリーミングは.NET Server？

昨年末，マイクロソフトから，Windows Mediaによるインターネット TVのサポートと，インターネットTV配信メーカーからのWindows Media 9形式による放送開始の発表などが行われました．筆者がもっとも期待しているのは5.1chによる映画配信などです．筆者の環境では，ワイヤレスバーチャルドルビーサラウンドヘッドホン以外に，従来のドルビーサラウンドTHX対応AVアンプによるホームシアター環境もありますが，DVDを買いに行く時間も少なく，またレンタル店に寄る時間も少ないためになかなか利用できないでいます．そこで，インターネットで映画がDVD並みの画質と音質で見られるのであれば，どんどん利用していきたいと考えています．DVD並みの画質でストリーミングが得られるメリットはとても大きいと思います．

● ビデオ画質の向上

音質ばかりに目がいきがちなのですが，映像品質も同ビットレートで従来のWM8と比べて格段に向上しているようです．従来のWM8形式のビデオをWindows Media 9で再生してみた場合でも，拡大表示などを行ったときの処理が良くなっているようでした．しかしながら，同一のアナログソースでも，Windows Media 9形式で圧縮したものを再生したときのほうが，WM8形式と同一ビットレートにおいて格段に鮮明さが異なっていました．とくに，1.5Mbps以上のビットレートではっきりとその差が出てくるようです．

さらに，高精細度（HD）ビデオ画像形式のサポートなど，音質面以外でも高度な画質と高ビットレートを活かした映像のサポートがなされています．

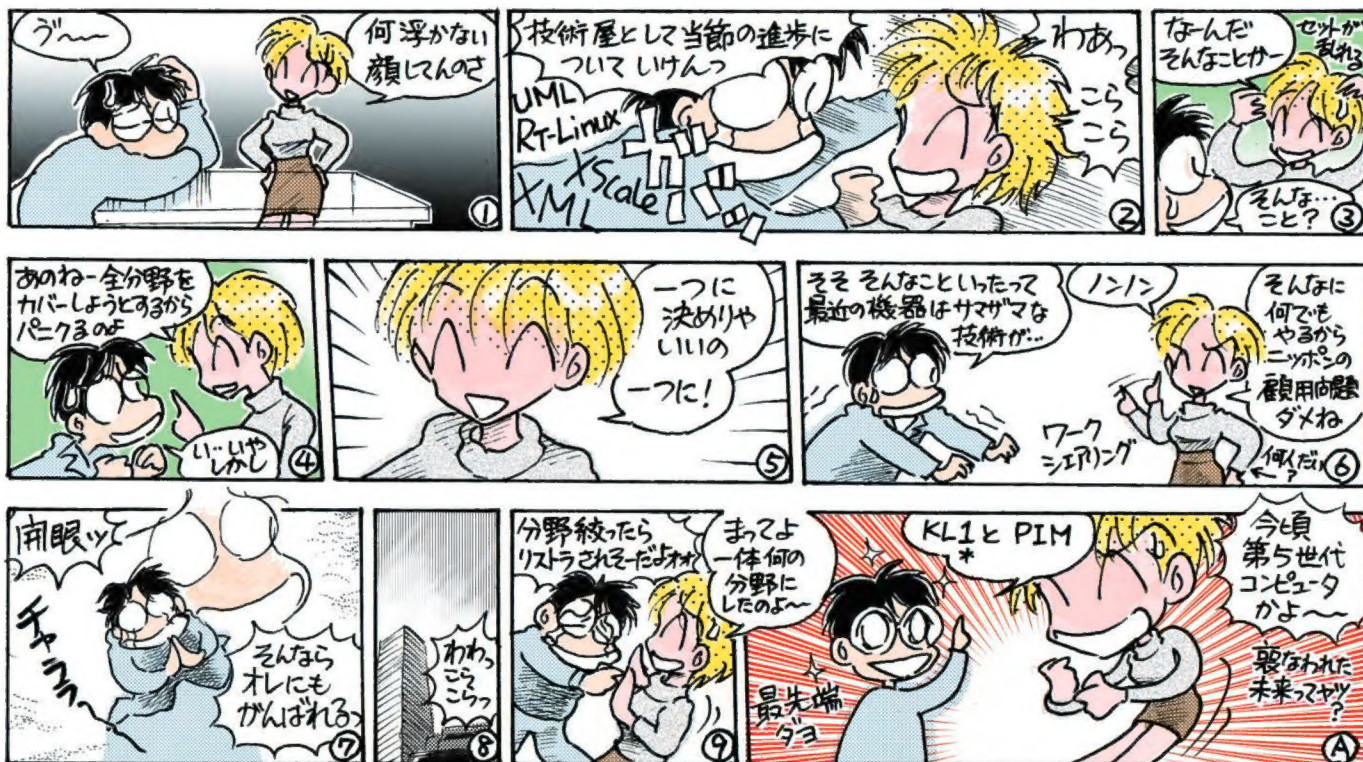
〔図〕Windows MediaのWebサイト

● Windows Media 9向け開発環境（SDK）

Windows Media 9エンコーダなどのサポートだけでは足りないときに，より密接なソフトウェア開発を可能にするWindows Media 9 series SDKも配布されています．個人で使用する範囲における通常の圧縮作業などにはWindows Media 9 Encorderがあれば十分でしょう．しかし，業務用途で独自のソフトウェアを構築するような場合には，ソフトウェア開発キットからの情報が役に立つことでしょう．

通常のアプリケーションを構築する場合は，DirectShowインターフェースを使用することで，Windows Mediaを含む多くのマルチメディアファイルフォーマットをサポートすることが可能です．DirectXのほうもVer9になり，Visual Studio.NETを使用したManaged DirectX（MDX）のサポートで，C#やVisual Basic.NETから多くのインターフェースが利用可能になっています．これらの組み合わせで，より高度な独自のアプリケーションの作成などへつながっていくことと思います．

ひろはた・ゆきお　OpenLab.

- Windows MediaのWebサイト（**図**）
  http://www.microsoft.com/japan/windows/windowsmedia/
- Windows.NET ServerにおけるWindows Mediaサービス
  http://www.microsoft.com/japan/windows/windowsmedia/services/net/default.asp

＊KL1：並列論理型言語　PIM：並列推論マシン

フジワラヒロタツの現場検証(69)

技術者生存戦略

最近，技術者にとってのシアワセとは，いったい何だろうかとシミジミ考えています．

二十代の頃は，とにかく「面白い仕事」がしたくてたまりませんでした．「面白い」というのはやはり先端的な，他の誰もやっていないような仕事です．インモス社のトランスピュータというチップなど，なかなかシビレました．さまざまなアーキテクチャとアイデアが，カンブリア紀の大爆発とはいいませんが多数出現していました．このころは，おもにハードウェアが舞台の脚光を浴びていたような気がします．ハードウェア能力はまだまだ低かったのですが，さまざまなアイデアがあちこちの方向に提案されていました(Pascalの中間コードを直接ハードウェアで実行しようとするPascalマイクロエンジンなど)．

今思うと，この頃の雰囲気は，高度経済成長期によく似ています．右肩上がりに成長する業界です．

しかし，インフラが整備されると，社会が成熟し，サービス産業など第3次産業に主役が移っていくように，コンピュータ業界も，大胆なハードウェアの変革の時期は終わり，ソフトウェアが主役になっていきます．もちろん，まだまだハードウェアも右肩上がりではありましょうけれど，パソコンはATアーキテクチャに固まり，自由奔放なアーキテクチャはなかなか生まれません．

そんな流れに身をまかせているうちに，筆者の技術も，だいぶ古めかしいものになってしまってきたような気がします．年とともに新しい技術分野は拡大し，一人で追える分野がどんどん狭くなっています．さらに悪いことには，ある分野を追うための「質」だけではなくて「量」もどんどん増加しているような気がするのです．

記憶力が落ちた代わりに，今まで培ってきた技術を磨き上げ，完成の域に達するというある意味職人的な技術者には生きにくい状況に，加速度的になってきつつあるのではないでしょうか．

それではどうすれば良いのでしょう？　広く浅く，広範な技術知識を仕入れていても，いざ，ある技術のプロとしてすぐさま立ち上がるのは困難です．そのような生存戦略を立てて生き残っていく人は，プロジェクト管理とコミュニケーション，そしてお金のプロになってマネジメントのできる，管理者の色彩が強い技術者になっていくのでしょう．

狭く，深く技術をきわめたい向きは，自分の技術分野があとどのくらい食っていけるのか，冷静に評価し続けなければなりますまい．そして，その技術がすたれる前に，次の飯の種に速やかに移っていくことがなにより必要です．しかも，この戦略をとるためには，そのために会社を移っていくことが必要になるでしょう．なぜなら，技術の移り変わりと会社の方向が一致しているとはまったく限らないからです．ですから，このようなスペシャリスト技術者は自ずと，独立したコンサルタントとなっていくことでしょう．

さて，筆者はどの道を歩むべきか……．と，いうわけでシミジミと技術者のシアワセについてまたまた考えるのでした．

藤原弘達　(株)JFP　デバイスドライバエンジニア，漫画家

480Mbps対応USBターゲットからホストシステムの設計まで

解説！USB徹底活用技法

特集

インテル製のチップセットにUSB2.0が内蔵されて以来，デスクトップPCでは急速にUSB2.0が普及してきている．最近ではノートパソコンでもUSB2.0を搭載したものが登場している．ターゲットとなる周辺機器においては，すでにUSB2.0対応機器がかなりの数になっている．

USB1.1とUSB2.0の最大の違いは，やはりデータ転送性能にあるだろう．USB1.1のときのようなCPU転送を主としたシステム構成では，480Mbpsの高速転送性能はとても活かしきれない．そこで本特集では，10Mバイト/秒を超える転送レートを実現する，USB2.0ターゲットシステムの設計事例を解説する．

また最近では，PDAなどPC以外の機器にUSBホスト機能が搭載されるようになってきた．これまでPCに対してターゲットとして動作してきたPDAが，デジカメやプリンタを接続するためにUSBホストの機能を実装しはじめたのである．組み込み機器にUSBホスト機能を実装するという要求は，今後も増えていくと予想される．本特集では，組み込み向けのUSBホストコントローラやUSBプロトコルスタック/ミドルウェアなどについても解説する．

レガシーフリー宣言！——USBのすすめ

編集部

● USBのすすめ

USBとはユニバーサルシリアルバスの略で，キーボードやマウス，プリンタなど，あらゆる周辺機器を一つのインターフェースで接続することを目的とした規格です．

USBはシリアルバスなので信号線の本数が少なく，それゆえ扱いやすいケーブルで接続できます．また，たとえば一つのCOMポートには1台のモデムしか接続できませんが，USBはUSBハブを使って簡単にポートを増やすことができます．またSCSIではIDの設定が必要でしたが，USBではその必要がありません．

このような使い勝手の良さが，USBを普及させた大きな要因といえるでしょう．

現在では，USBを装備していないPCは皆無といってもよいでしょう．また，USBで接続できないPC周辺機器は，ディスプレイくらいのもので，それ以外はありとあらゆる周辺機器がUSBで接続可能になっています．

● 組み込み機器のUIとして

それ自身にキーボードや表示デバイスをもたない組み込み機器の場合，動作モードやパラメータを設定する必要があるときにはどうすればよいでしょうか．

数ビット分のモード切り替え程度であれば，ディップスイッチなどを実装する程度で済みますが，さまざまなモードやパラメータを必要とする場合は，何らかのユーザーインターフェースを実装して，メニューなどから設定させるようにするでしょう．

このようなインターフェースとして，これまではRS-232-Cを使ってターミナル機能を実装するのが一般的でした．RS-232-Cのような調歩同期式のシリアルコントローラは，単体プロセッサでもないかぎり，組み込み向けマイコンであればほぼ必ず内蔵されています．これにRS-232-CドライバICを接続するだけで，電気的なインターフェースは完成するので，実装が簡単なのです．

しかし現在では，COMポートをもたないノートパソコンが過半数を占めています．不具合の発生した組み込み機器に，現地調査でノートパソコンをもっていったが，COMポートがなくて接続できなかったという笑い話を，あなたも体験するかもしれません．

● USB2.0で高速転送

USBは，さらに通信速度を高速化したUSB2.0の登場で，HDDやDVDドライブなどのストレージデバイスの接続インターフェースとしても十分実用的になってきました．組み込み機器でも，この高速データ転送能力は魅力的です．

たとえば，ネットワークに接続する組み込み機器の場合，UIの切り口としてもEthernetを使うことがよくあります．telnetで接続してターミナルを実現する簡易的なものから，中でhttpサーバを走らせ，WWWブラウザから設定が行えるGUI対応の高機能なものまであります．

しかし，100Base-TのEthernetで10Mバイト/秒を超えるデータ転送は実現できません．10Mバイト/秒を超えるような高速データ転送を必要とするなら，USB2.0が最適です．

とくにネットワークに接続する必要のない機器であれば，PCとの接続インターフェースとして，現在ではUSBを採用するのが最適でしょう．

● USBとIEEE1394

USB1.1の頃は，USBとIEEE1394を比較した場合，IEEE 1394には高速データ転送が可能という利点がありました．しかし，USB2.0の登場によって，この差はほとんどなくなったといえるでしょう．とはいっても，USBが高速化された現在でも，この両者は決定的に異なる点があります．

IEEE1394には，基本的にホストやターゲットという区別はありません．すべての機器が対等で，ポイントtoポイントで接続することが可能です．そして，必要があれば自分からデータ通信を開始することができます．

USBはホストとターゲットが明確に区別されていて，頂点に立つホストは，そのツリーの中でただ一つしか存在できません．また，ターゲットは自分からデータ通信を開始することはできません．あくまでホストからの問い合わせに応えるというプロトコル構造になっているのです．

以上のような違いから，USBはホストであるPCを中心に，キーボードやマウス，プリンタやデジカメといったPC周辺機器を接続するインターフェースとして急速に普及しているのです．

● USB On-The-Go

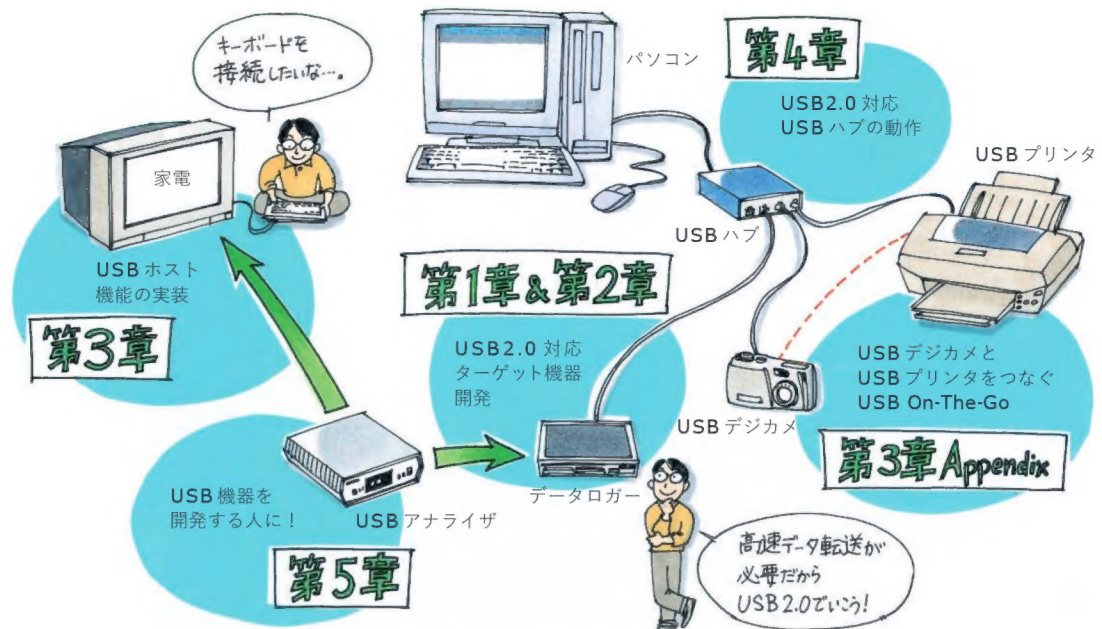
ここにUSB対応のプリンタとデジカメがあったとします．そこにPCがあれば，PCをホストとしてプリンタとデジカメをターゲットとして接続します．デジカメで撮った写真はいったんPCに転送し，それからプリンタで印刷します．

しかし，そこにPCがなかったとしたらどうでしょうか．デジカメはあくまでUSBターゲットなので，USBプリンタと直接接続することができません．つまり，デジカメで撮った写真を印刷することができないのです．

そこで，USBターゲット機器に一時的にUSBホストの機能を持たせ，USB周辺機器同士を直接接続できるようにしたのがUSB On-The-Goという規格です．

● 特集案内

第1章は，480Mbpsのハイスピードに対応したUSBターゲットコントローラEZ-USB FX2（サイプレス）の詳細について解説

〔イラスト〕
特集で解説する分野

します．スレーブFIFOとGPIFがFX2の性能を活かすカギとなります．そして第2章では，FX2を使った高速データ転送対応USB機器の設計事例を解説します．事例では10Mバイト/秒を超えるデータ転送性能を実現しています．

第3章では，組み込み機器にUSBホスト機能を実装するために，OHCIに準拠したUSBホストコントローラを内蔵したSH7727(日立)を取り上げ，さらに市販のUSBホスト対応プロトコルスタックの移植事例を解説します．

また，第3章Appendixでは，USB On-The-Go対応デバイスISP1362(フィリップス)について紹介します．また，SH7727のようなUSBホストを使ったほうがよいのか，ISP1362のようなOn-The-Go対応デバイスを使ったほうがよいのかも考察しています．

さらに第4章では，USB2.0対応のハブコントローラの内部動作について解説します．今後はハブコントローラの違いにより性能に差が出てくるかもしれません．

最後に第5章では，これらUSB対応機器を開発する場合に力強い味方となる，USBアナライザの活用事例について解説します．

コラム　USB規格とUSB1.1対応コントローラの使い方

本特集ではUSB2.0に対応したターゲットコントローラとしてEZ-USB FX2を取り上げていますが，USB1.1対応のターゲットコントローラの活用法や，Windows/Linux用のUSBドライバの作成事例などについては『USBハード&ソフト開発のすべて』に詳しい解説があります．

- 第1章　USBデータ転送プロトコルの基礎知識
- 第2章　USBコントローラML60851Dを使ったUSBターゲット機器の開発
- 第3章　USBコントローラUSBN9602を使ったUSBターゲット機器の開発
- 第4章　USBコントローラ内蔵マイコンAm186CUを使ったUSBターゲット機器の開発
  Appendix　USBアナライザが役立った
- 第5章　ワンチップマイコン内蔵USBコントローラAN2131SCを使ったUSBターゲット機器の開発
- 第6章　WDMの基礎とUSBドライバの構造
- 第7章　ML60851D評価ボード用テストドライバの作成
- 第8章　汎用USBドライバの使い方と内部構造
- 第9章　汎用USBドライバ活用法
- 第10章　Windows2000用USBシリアルポートドライバの作成
- 第11章　Windows98用USBシリアルポートドライバの作成
- 第12章　Windows2000用NDIS/WDMデバイスドライバの開発
- 第13章　Linux用USBドライバの作成事例

TECH I Vol.8
USBハード&ソフト開発のすべて
USBコントローラの使い方から
Windows/Linuxドライバの作成まで

B5判280ページ(CD-ROM付き)
定価2,200円(税込み)
ISBN 4-7898-3319-4

Chapter 1

スレーブFIFOやGPIFを搭載した高性能USBターゲットコントローラ

USB2.0対応コントローラ EZ-USB FX2の詳細

桑野雅彦

本章ではまず，ハイスピード(480Mbps)対応のUSBターゲットコントローラEZ-USB FX2の詳細を解説する．とくに高速転送を実現するために実装されたスレーブFIFOおよびGPIF(General Programmable Interface)機能について，重点的に解説する．スレーブFIFOとGPIFを駆使することで，CPUを介さずに高速なデータ転送を実現できる．

（編集部）

1 インテリジェント型ターゲットコントローラEZ-USB FX2

1.1 FX2の概要

Cypress社のCY7C68013(EZ-USB FX2：以下FX2と略)は，従来のUSB1.1対応のUSBターゲットコントローラであるEZ-USBファミリのコンセプトを継承しながら，USB2.0対応にしたデバイスです．

FXシリーズに代表されるEZ-USBファミリは，CPUコアとしてタイマやカウンタ，シリアルポート，割り込みコントローラなどを内蔵した8051コアをベースとして，次のような機能が拡張されています．

- CPUによる連続転送に便利なオートポインタ機能の追加
- 2個目のUARTの追加
- 16ビットタイマを1本(TIMER2)追加
- マルチプレクスされていない高速外部バス
- 8個の割り込み拡張(INT2～INT5，PFI，T2，UART1)
- 可変外部バスタイミング機能

さらに，外部I/O，I²Cバスインターフェース，8Kバイト(品種によっては4Kバイト)のプログラム/データ用RAMなどを内蔵しています．USBインターフェース部分も，大きくランダムアクセス可能なエンドポイントバッファ，シリアルROMやUSBバス経由でのファームウェアダウンロード機能などをもつ，非常にユニークなデバイスです．

〔写真1〕**EZ-USB FX2**(CY7C68013)**の外観**

FX2はFXシリーズのもつこれらの特徴を受け継ぎながら，CPUクロックを24MHzから48MHzに引き上げ，さらにUSB2.0の高速な伝送に対応したものです(**写真1**)．

1.2 FX2の内部ブロック

FX2の内部ブロックは**図1**のようになっています．USB2.0対応のトランシーバ，CPUコア，8.5Kバイトのメモリ，4KバイトのエンドポイントFIFOなどに加えて，GPIFと呼ばれる簡易ステートマシンや，I²Cコントローラなどが内蔵されています．次に，各ブロックについて見ていきましょう．

● USBトランシーバ

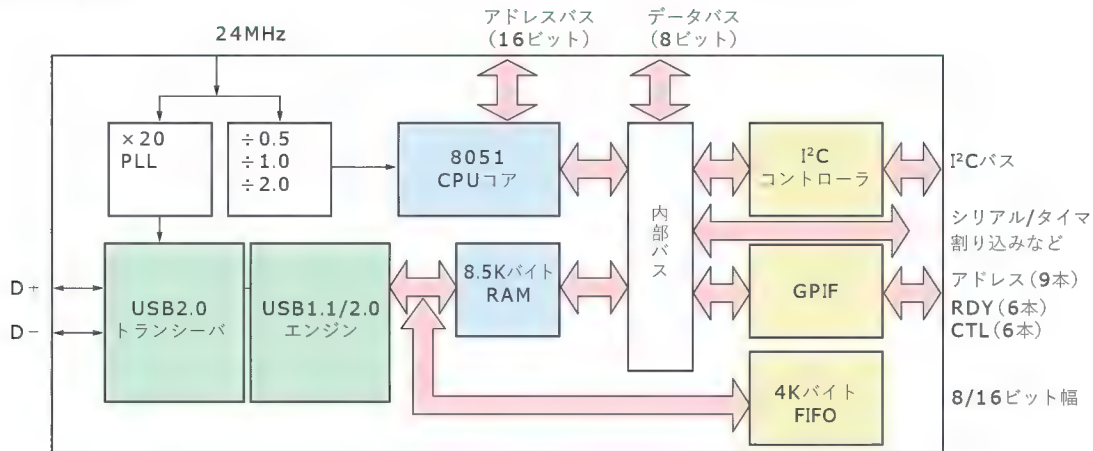
USB2.0はUSB1.1の上位互換なので，USB2.0対応のハイスピード(480Mbps)のデバイスは，ホストがUSB1.1ならばUSB1.1のフルスピード(12Mbps)デバイスとして動作します．このため，FX2内部にも1.1(フルスピード)対応のトランシーバと2.0(ハイスピード)対応のトランシーバの両方が内蔵されています．

● USBコントローラ

USBコントローラは当然のことながら，USB1.1のフルスピードとUSB2.0のハイスピードの両方をサポートしています．

FXシリーズと同様に，FX2もCPUのリセット制御や，内蔵SRAMへのアクセスなどの権利の主導権はUSBコントローラ側が握っています．またUSBコントローラが8051の助けを借りることなく，USBデバイスとして動作可能であるという点もFXシリーズと同様の大きな特徴です．

ブート手順もFXシリーズと同様で，USBコントローラがシリアルROMの先頭バイトをチェックし，IDに応じた動作を行います．

〔図1〕FX2のブロック図

▶先頭バイトがC0h

このときにはシリアルROMから，ベンダID，プロダクトID，デバイスIDがシリアルROMに格納されているので，それぞれシリアルROMから読み出して，そのIDを使って起動します．CPU(8051)はリセットは解除されませんが，USBコントローラが単独でUSBデバイスリクエスト処理を行うので，ホスト側からは通常のUSBデバイスとして認識されます．

▶先頭バイトがC2h

このとき，シリアルROMには8051が実行するためのプログラムが格納されます．ベンダIDやデバイスIDなども同じように格納される領域は決められているのですが，これらの値は実際には使われません．

シリアルROMの内容はデバイス内部のSRAM(全部で8.5Kバイト)にダウンロードされ，格納が終了するとCPUのリセットが解除され，CPUはSRAM上に展開されたプログラムを実行します．

▶先頭バイトがそれ以外，またはシリアルROMが存在しない

USBコントローラは，デフォルトのチップベンダのIDを使って起動します．

- ベンダID：04B4h
- プロダクトID：8613h
- デバイスID：(デバイスのバージョン番号)

チップベンダから提供されているFX2開発サポートツールであるEZ-USBコントロールパネルなどを使うためには，このIDで起動している必要があります．

● 8051CPUコア

FX2内蔵の8051コアは，最高48MHzで動作します．命令コードはオリジナルの8051と完全に互換性がありますが，1マシンサイクルがオリジナルでは12クロックを要するのに対して，FX2内蔵CPUコアは4クロックと1/3で済んでいる点が異なります．

命令実行に必要なマシンサイクル数は数値演算命令(DIV，MULなどは除く)，論理演算，データ転送など，多くの命令でが命令バイト数と一致しています．つまり1バイト命令は1マシンサイクル(4クロック)で，2バイト命令は2マシンサイクル(8クロック)かかるものが大半です．

● 内蔵RAM

FX2のメモリマップは**図2**のようになっています．FX2には8.5KバイトのSRAMが内蔵されていますが，このうち0.5KバイトはE000h～E1FFhの領域で，データ専用，残り8Kバイト分はメインRAMと呼ばれ，データとプログラム領域の両方に使うことができます．

メインRAM領域は，1FFFH番地以下の部分に配置されています．ただし，8051ファミリCPUの特徴として，汎用レジスタやフラグ類，特殊レジスタ類がメモリ空間の下位256バイトまでの領域(0000h～00FFh)に配置されるので，この領域の取り扱いには注意が必要です．

とくに8051の場合，0080h～00FFhの領域はSFR(スペシャルファンクションレジスタ)と汎用データメモリで共用されており，どちらをアクセスするかはアドレッシングモードによって切り替わるという，多少変わったアーキテクチャになっています．8051対応のCコンパイラでも，この点には配慮されていて，たとえば_sfrのようなキーワードを変数宣言の先頭に付けることで，SFR領域へのアクセスを行うようなコード生成がなされます．

● I²Cコントローラ

I²CバスはFX2が起動時に動作モードを決めたり，プログラムを読み込んで内蔵SRAMにプログラムを転送するためのシリアルROM(ブート用シリアルROM)を接続するほか，市販のI²Cバスデバイスを接続する目的にも使用可能です．

I²Cバスではアドレスが3ビットあるので，論理上は8デバイスまで接続可能ということになりますが，このうちブート用のシリアルROMはアドレス001hを使用しています．これ以外のアドレスはユーザー側で任意に使うことができます．

ちなみに，チップベンダが提供しているFX2デベロップメントキットではPCF8574(Philips)というI²Cバス対応のディジタル入出力デバイスを使ってLEDやスイッチを接続することで，汎用I/Oポートを使わずに実現しています．

〔図2〕FX2のメモリマップ

アドレス	内　容	サイズ
FFFFh ～ FC00h	EP8バッファ	1024バイト
FBFFh ～ F800h	EP6バッファ	1024バイト
F7FFh ～ F400h	EP4バッファ	1024バイト
F3FFh ～ F000h	EP2バッファ	1024バイト
EFFFh ～ E800h	(予約済み)	
E7FFh ～ E7C0h	EP1IN	64バイト
E7BFh ～ E780h	EP1OUT	64バイト
E77Fh ～ E740h	EP0 IN/OUT	64バイト
E73Fh ～ E700h	UNAVAILABLE	64バイト
E6FFh ～ E600h	(各種レジスタ)	256バイト
E5FFh ～ E480h	(予約済み)	384バイト
E47Fh ～ E400h	GPIF波形テーブル	128バイト
E3FFh ～ E200h	(予約済み)	512バイト
E1FFh ～ E000h	データRAM	512バイト
1FFFh ～ 0100h	汎用メモリ	8Kバイト
00FFh ～ 0080h	汎用メモリ/SFR	
007Fh ～ 0030h	汎用メモリ	
002Fh ～ 0020h	Bit-Addressable RAM	
001Fh ～ 0018h	レジスタバンク#3	
0017h ～ 0010h	レジスタバンク#2	
000Fh ～ 0008h	レジスタバンク#1	
0007h ～ 0000h	レジスタバンク#0	

1.3　エンドポイント構成

FX2はUSB2.0で拡張された480Mbpsという高速な伝送に対応するために，エンドポイントの構成がFXシリーズから変更されています．フルスピードモード(12Mbps)では，バルク伝送のパケットサイズが最大64バイトであったのに対して，USB2.0のハイスピードモード(480Mbps)では，最大パケットサイズが512バイトまで拡張されました．これに対応して，FX2でもエンドポイントバッファの構成が大幅に変更されました．

● ダブルバッファ構造のEP2/4/6/8

FXシリーズではユーザーが自由に扱えるエンドポイントはバルク/インタラプト伝送用のものがIN/OUT方向それぞれについて64バイト×7本(INEP1～INEP7，OUTEP1～OUTEP7)の計14本，アイソクロナス伝送用エンドポイントがIN/OUTそれぞれ8本の計16本(バッファ領域は2Kバイトのアイソクロナス用メモリをシェアして利用)の合計30本のユーザー用のエンドポイントをもっていました．

これに対しFX2では，エンドポイントバッファの構成は64バイトのINEP1/OUTEP1，および512バイト×2バンクのダブルバッファになったエンドポイントを4本(EP2，EP4，EP6，EP8)もつようになっています．FXシリーズでは隣のエンドポイントとペアリングすることでダブルバッファを実現していましたが，FX2ではEP2/4/6/8はデフォルトでダブルバッファになっており，コンフィグレーションによって，トリプル(3バンク)バッファやクワッド(4バンク)バッファにもなりますが，シングルバッファにすることはできません．FX2で設定可能なエンドポイントバッファ構成を**図3**に示します．

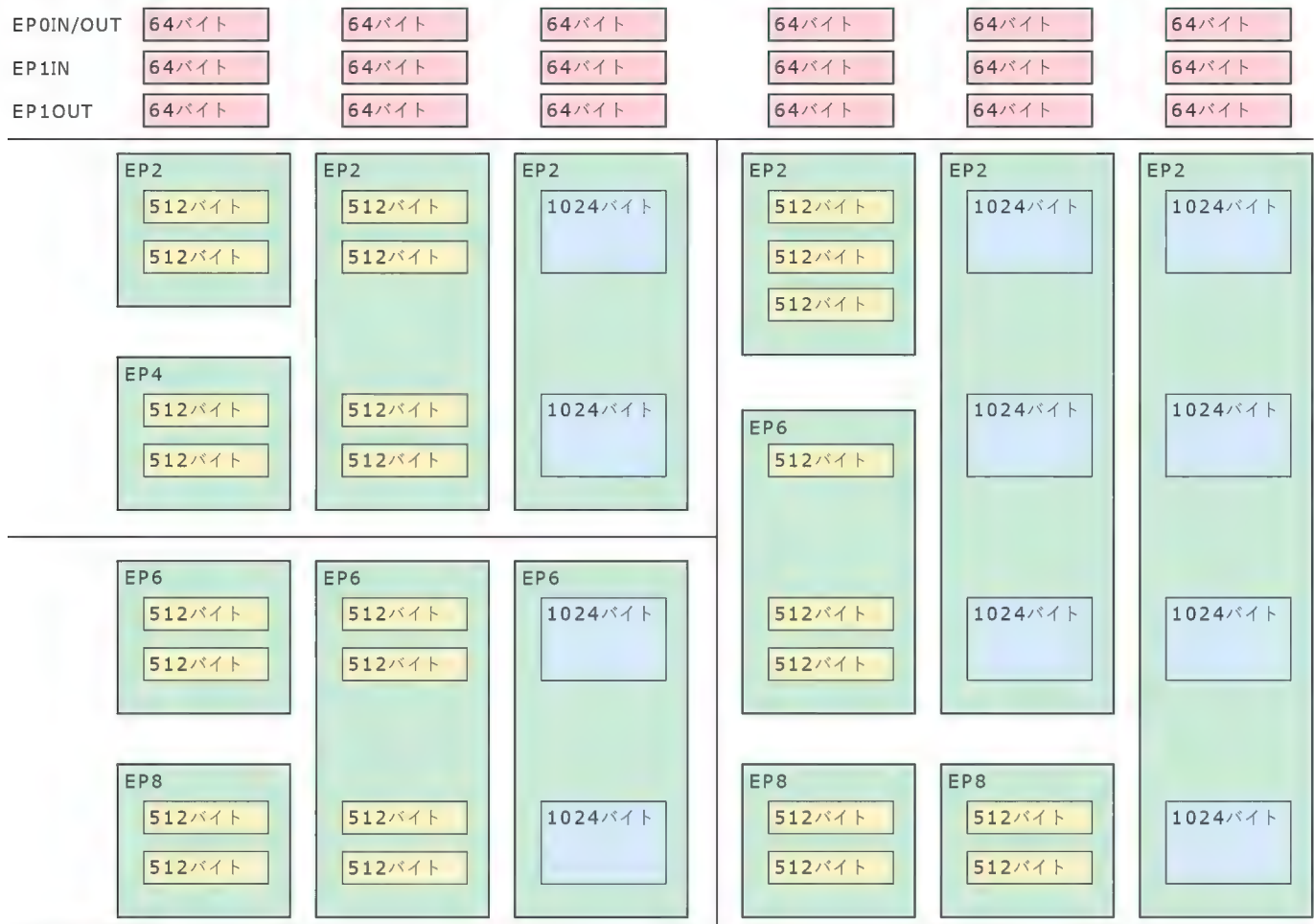
● エンドポイントの組み合わせ

図3の左側が上下に分かれている部分は，その範囲内で上下を任意に組み合わせられることを示し，右側はその組み合わせ以外はできないことを示しています．たとえば，EP2とEP4をそれぞれ512バイトのダブルバッファ(左端の組み合わせ)，EP6を1024バイトのダブルバッファ(左から3番目)として使うということは可能ですが，EP2を512バイトのトリプルバッファとして使う場合(右から3番目)は，ほかのエンドポイントは512バイトのトリプルバッファのEP6，512バイトのダブルバッファのEP8に自動的になるということです．

なお，EP1IN，EP1OUTは，バルク，インタラプト，アイソクロナスのいずれでも使用することができますが，バッファサイズは64バイトでシングルバッファで，後述するようなGPIFによる高速なやりとりに使うこともできないので，もっぱら少量の補助的なデータのやりとりやステータスの取得などに使うのに向いているといえるでしょう．

● ダブルバッファ時の切り替え

ダブルバッファの切り替え動作はUSBコントローラが自動的に行います．また，どちらのバッファもまったく同じアドレスへのアクセスになるので，ソフトウェア側で切り替え動作が行わ

〔図3〕FX2のエンドポイントの構成

れていることを意識する必要はありません．たとえば，OUT方向(ホストからターゲットへの伝送)のエンドポイントの場合ならば，立て続けにホストからデータが来る場合，最初に1パケット分のデータが到達し，CPUがバッファの内容を処理している間に次のパケットのデータがもう片方のバッファに格納されます．CPUが最初のパケットの処理を終えてエンドポイントのアクセス権をUSBコントローラ側に渡したとき，すでにもう一つのバッファに1パケット分のデータが入っていれば，CPUがアクセス権を渡した直後に新しいパケットがホストから来たかのように見えるわけです．

1.4 スレーブFIFOとGPIF

● CPU転送では間に合わない

FX2の前身にあたるFXシリーズと同様にFX2もCPUコアを内蔵しており，このCPUによってすべての処理を行うということも不可能ではありませんが，これではせっかくの高速なデータ伝送が活かせません．USB1.1ではバス上の最大伝送速度が12Mbpsだったので，FXシリーズではエンドポイントバッファと外部との間のデータ転送はすべてCPUが行う設計になっていましたが，同じことをFX2で行うと，単に1パケットのサイズが大きくなったUSB1.1のようになってしまいます．USB2.0を使いたい場面というのは，パケットサイズの拡大よりも速度の大幅な向上が求められている場合が多いというのに，これではまったく意味がありません．

CPUを高速化して対処するということも不可能ではありませんが，消費電力の上昇やコストアップにつながります．また，USB2.0を利用したい場面というのは，どちらかといえば，FX2をUSBインターフェースチップとしてとらえ，外部にパラレルバスを付けてデータの入出力を直接行うという場合が多いと思われますが，このような単純なデータ転送のためにCPUを使うというのは得策とはいえません．

DMAコントローラによる方法も，CPUバスを占有するようでは，大量のデータ伝送が発生したときにはDMAの転送動作によってCPUバスがロックしてしまうことになります．

● スレーブFIFOとGPIF

このような事情に配慮して，FX2ではより簡単でかつ巧妙な方法として，USBのエンドポイントバッファがそのまま外部との間のFIFOバッファメモリに切り替わるスレーブFIFO機能

をもたせ，さらにそして周辺回路やデバイスがノンインテリジェント型である場合に，スレーブFIFOと周辺回路の間のデータ伝送制御をCPUやバスインターフェースユニットに成り代わって行うGPIF(General Programmable Interface)を組み込んでいます．このスレーブFIFOとGPIFがFX2の要(かなめ)ともいえる機能で，これらをいかに利用するかによってパフォーマンスが大きく変わってきます．

ここでは，FX2の大きな特徴であるスレーブFIFOとGPIFに焦点をあてて説明し，最後に実際にスレーブFIFOとGPIFを使ったデータ転送を行ってみることにします．

FX2のCPUコアまわりなどの一般的な機能などについては，チップベンダのデータシートや従来のEZ-USBの説明などを参考にしてください．

1.5 FX2のパッケージ

FX2(CY7C68013)はデータブックによると，56ピン，100ピン，128ピンの3種類のパッケージが存在します．ピン数が多いものは少ないものの信号をすべてもっています．これらの信号を整理したのが**図4**です．

〔図4〕FX2のパッケージと信号群の関係

● 56ピンパッケージ

56ピンパッケージは8ビットのI/Oポートを三つ(PORTA，PORTB，PORTD)とI^2Cバス，GPIF/スレーブFIFOコントロール信号などが引き出されています．

このうち，PORTBとPORTDは16ビット(8ビットとしても使用可能)のFIFOデータバスとして使用できるようになっています．後で説明しますが，GPIFとスレーブFIFO機能はFX2を他のUSBコントローラと大きく差別化しているポイントで，最小限の外付け回路でCPU転送では不可能なほどの高速伝送を実現することが可能となっています．

このもっとも小さいパッケージのFX2を一つ使いGPIFとFIFO機能が連動することで，外部にPLDなどのグルーロジックはおろか，バッファすら使わずにIDE/ATAPIインターフェースとのブリッジ機能が実現可能です．それも，PIOモードだけではなく，UDMA/66での動作が可能であるという点は特筆すべきところでしょう．

実際にチップベンダ純正のデベロップメントキットや，参考文献2)で紹介したFX2評価ボードでも40ピンのIDEコネクタが用意されており，USB-IDE/ATAPI変換アダプタとして動作させることができるようになっています．これらのボードに使われているものは実際には128ピンパッケージですが，IDEコネクタに引き出されている信号はすべて56ピンパッケージでも引き出されている信号なので，56ピンパッケージ品1個で十分実現可能です．

● 100ピンパッケージ

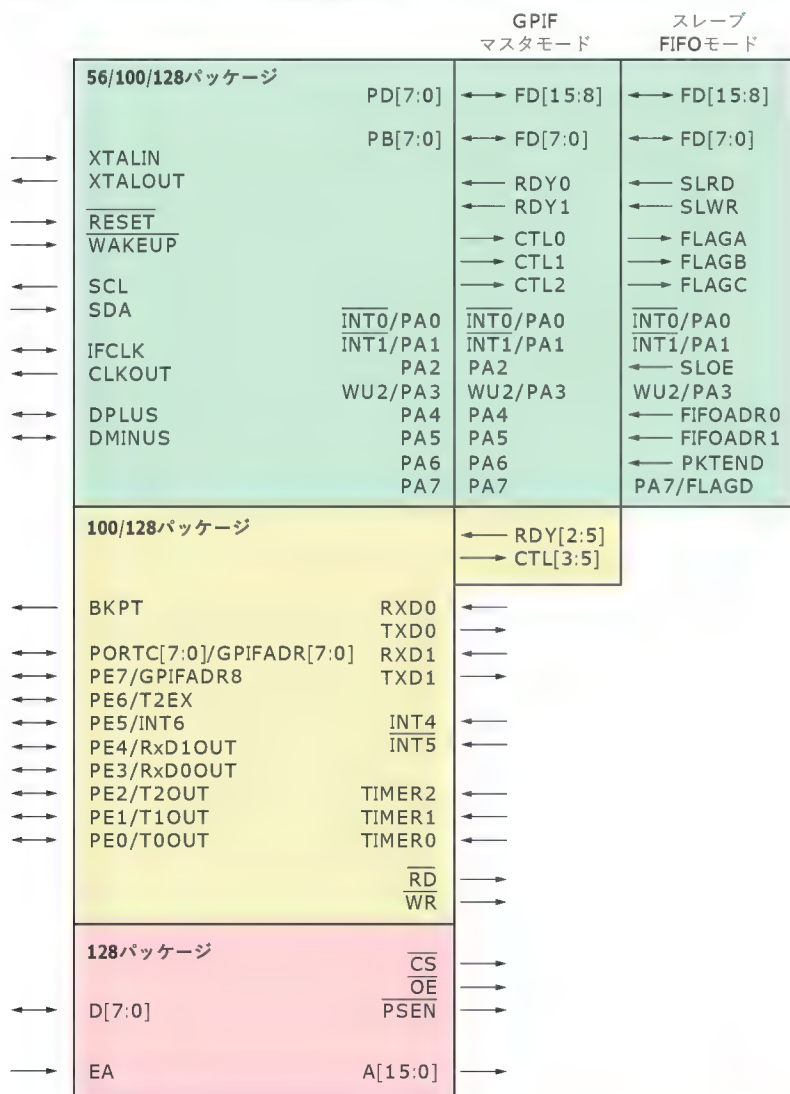
100ピンパッケージでは，56ピンパッケージに加えて，次のような信号が引き出されます．

- 8ビットI/Oポート×2(PORTC，およびPORTE)
- GPIF/スレーブFIFO信号×9
- 2チャネルシリアルポート
- 3チャネルタイマ入力
- 割り込み信号×2(INT4，INT5)
- PORTC用のRD/WRストローブ信号

PORTCとPORTEにはGPIF用のアドレスやタイマ，シリアルポート関係の信号がマルチプレクスされています．

● 128ピンパッケージ

128ピンパッケージでは100ピンパッケージに加えてFX2の内蔵CPUコア，8051の外部バスが出力されます．16ビットのアドレスバス，8ビットのデータバス，アドレス/データのコントロール信号やチップセレクト信号などが追加されています．

100ピンパッケージは0.65mmピッチのTQFPです

が，128ピンパッケージは0.5mmピッチのTQFPなので，ピン数は増えてはいますが，基板上の占有面積はほとんどかわりません．価格差がどの程度あるのかはわかりませんが，これから基板を新しく起こすのであれば，128ピンパッケージのほうが拡張性も高く便利であるように思えます．

2 スレーブFIFO

以降では，480Mbpsの高速転送に対応すべく実装された各機能について，重点的に解説していきます．まずはスレーブFIFOです．

2.1 エンドポイントバッファとスレーブFIFO

● 従来のエンドポイントバッファ

USBターゲットデバイスコントローラのエンドポイントバッファは通常，セマフォフラグ付きのメモリブロックとして実装されています．

たとえばOUT方向(ホストからターゲット)のエンドポイントならば，次のような手順になります．

(1) ホスト側から送られてきたデータはターゲットのUSB SIE(シリアルインターフェースエンジン)によって受信され，バッファメモリの先頭番地から順次格納される
(2) 1パケット分のデータを受け取ると，SIEによってデータ到達のフラグやデータ数のカウンタがセットされる(割り込みの発生がイネーブルになっていれば割り込みが発生)
(3) ターゲット側のCPUはこのフラグや割り込みによってデータ到達とデータバイト数がわかるので，バッファメモリからデータを取り出す
(4) 取り込みが終わった後，CPUは特定のポートをアクセスするなどして，SIEに対してエンドポイントのアクセス権を譲渡する
(5) SIEは再びホストからのデータをこのエンドポイントバッファに格納する

これを図示したのが**図5**です．FX2の前身でもあるFXシリーズでもこれと同じような実装が行われていて，エンドポイントバッファに入ったデータはCPUによって取り出され，処理が終わった後はCPUがバイトカウントレジスタに書き込みを行うことで，エンドポイントバッファのアクセス権譲渡を行っていました．

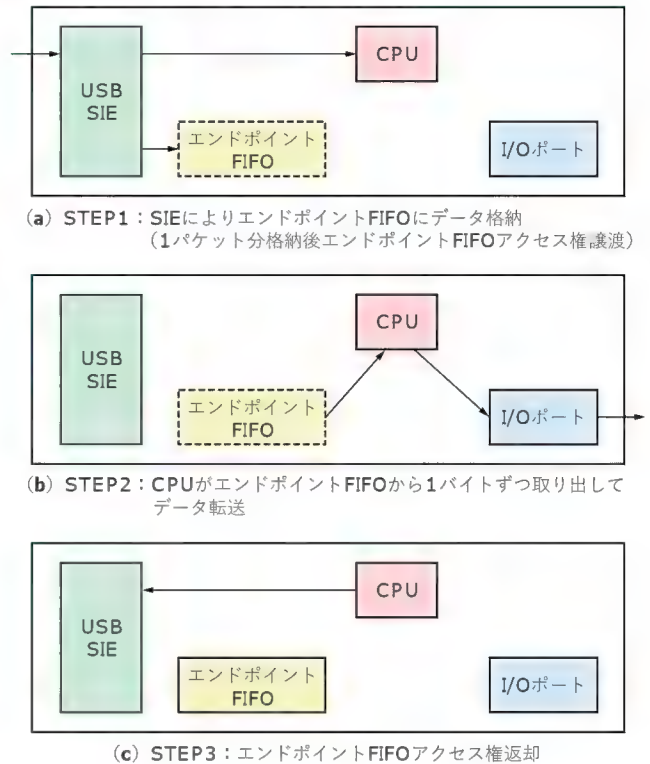
IN方向も向きが変わるだけで同様の考えで設計され，エンドポイントバッファへのデータセットのあとアクセス権をSIEに渡すとホストのIN要求に対してSIEが応答し，ホストに送り終わると完了フラグが立つというのが一般的な実装です．

USB1.1の最大12Mbps程度の伝送速度であれば，このような方法でもたいていの用途で十分だったのですが，USB2.0で最大の売り文句である，ハイスピードモード(480Mbps)の高速なデータ伝送にこの方法で対応するのは非常に難しくなります．

〔図5〕従来のUSB伝送動作(OUT方向)

(a) STEP1：SIEによりエンドポイントFIFOにデータ格納
(1パケット分格納後エンドポイントFIFOアクセス権譲渡)

(b) STEP2：CPUがエンドポイントFIFOから1バイトずつ取り出してデータ転送

(c) STEP3：エンドポイントFIFOアクセス権返却

● 高速CPUやDMAを使う？

FX2に内蔵されているCPUコアは8ビットの8051互換コアであり，動作クロックも12MHz(48MHzで4クロックが1サイクルのため)にすぎません．1回の転送に1クロック命令を6個使うだけで2Mバイト/秒(16Mbps)が限界ということになってしまいます．

CPUによる転送では，間にあわないような場合によく使われるのはDMAです．FXシリーズでも内部にデータ転送専用のDMAコントローラを内蔵していました．しかし，一般的にDMAはデータ転送動作にCPUバスを利用するため，大量のデータ伝送が頻繁に行われる場合には，DMAがバスを占拠してしまい，CPUの身動きがとれなくなってしまいます．また，DMAによる伝送制御は設定や取り扱いが複雑であったり，データ転送制御の自由度が低く，単純なデータ伝送だけではすまない場合には対処しきれません．

● FX2でのエンドポイントバッファ

FX2ではこのような問題に配慮し，よりシンプルで便利な方法として，エンドポイントバッファが周辺デバイスとの間のFIFOメモリに切り替わるというしかけを用意しています．このエンドポイントバッファと切り替わるFIFOのことを，スレーブFIFOと呼んでいます．

エンドポイントバッファがスレーブFIFOになるということは少し想像しにくいかもしれないので，少し説明を補足しておきましょう．

エンドポイントバッファは，通常USBのインターフェースコントローラ(SIE)の管理下にあり，CPUとの間で排他的なアクセスを可能としています．もちろん，FX2でもこれと同じような動作モードもありますが，FX2ではこのモードに加えて，エンドポイントバッファのアクセス権がSIEから離れたとき(OUT方向なら，データが1パケット詰められた後の状態)に，エンドポイントのデータバスや，アドレス，リード/ライトコントロール信号やFULL/EMPTYフラグなどがFX2の外部I/Oピンに接続されるというモードを用意したのです．

たとえばOUT方向であれば，SIEがデータを受け取った後，スレーブFIFOへの切り替えを行うだけで，FX2内部のCPUコアを介在せずに外部回路が直接エンドポイントバッファに入れられたデータを引き取ることができるようになります．外部回路から見ると，FX2に内蔵されたFIFOメモリに自動的にUSBのパケットデータが詰め込まれ，フラグが動作するようになるわけです．従来のようなCPUによるオーバヘッドもなくなり，大幅な性能改善が図れるというわけです．

FX2では，このスレーブFIFOとの切り替わりを自動的に行うAUTOモードと，CPUがレジスタ操作で切り替わりを行うモードを用意しています．後者のモードにすると，CPUによって不要なパケットを削除したり，INデータのサイズを変更したり，データの中身を変更することも最小限のオーバヘッド増加だけで行えるようになっています．スレーブFIFO機能を使った伝送動作の考え方を**図6**に示します．

スレーブFIFOのデータバス幅は8ビットモードと16ビットモードを選択できるようになっています．さらに，スレーブFIFOは最高48MHzでの動作が可能なので，理論上の最大転送速度は96Mバイト/秒となり，USBのバス速度(480Mbps＝60Mバイト/秒)をも上まわります．

2.2　スレーブFIFOの二つの動作モード

先ほど少し触れたように，スレーブFIFOは大きくわけて二つの動作モードがあります．一つはCPUがUSBエンドポイントバッファとスレーブFIFOを切り替えるモード，もう一つはCPUを介さずに自動的に切り替わるモードで，AUTO-IN/AUTO-OUTモードと呼んでいます．**図7**にパケットの流れのイメージをまとめてみました．

AUTO-IN/OUTを使わない場合，1パケット分のデータが到達する度にCPUが関与して切り替え操作(といっても，レジスタ一つか二つに書き込むだけだが)を行う必要がありますが，たとえばOUT方向であれば，受信されたパケットの中身やデータサイズをチェックしたり，スレーブFIFOに切り替えずに中身を破棄し，そのままSIEにエンドポイントバッファを返す(マニュアルでは‘SKIP’という単語を使っている)ことや，IN方向であれば，データの中身やホストに転送するサイズを変更するといった細工が可能です．

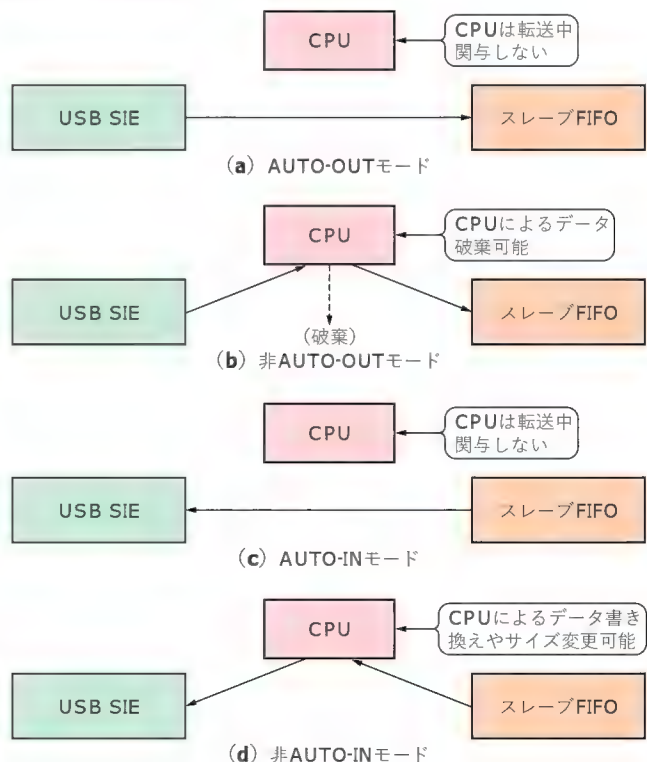
CPUが関与する分だけ若干オーバヘッドがかかりますが，1パケット(USB2.0のバルク転送では512バイト)に1回の処理で

〔図6〕FX2のスレーブFIFO機能を使った伝送例(OUT方向)

(a) STEP1：SIEによりエンドポイントFIFOにデータ格納
(1パケット分格納後エンドポイントFIFOアクセス権を譲渡)

(b) STEP2：エンドポイントFIFOがスレーブFIFOに切り替わる
(外部回路がFIFOとデータをやりとりする)

(c) STEP3：FIFOが空になると再びエンドポイントFIFOに戻る

〔図7〕AUTO-IN/OUTと非AUTO-IN/OUT

よいことと，FX2 のエンドポイントバッファがダブルバッファ（またはそれ以上）であるため，CPU によるパケットデータのチェックなどの処理の大半を伝送時間の部分に埋め込むことが可能であることから，性能低下は最小限に抑えられます．

2.3 スレーブ FIFO 専用バス

エンドポイントバッファがスレーブ FIFO に切り替わって動作しているとき，データバスやコントロール信号は CPU 内部バスとは完全に独立しており，外部のデータ転送によって CPU 側が影響を受けることはありません．

外部バスもまた，CPU バス信号とは別の PIO（Programmable I/O）ピンの一部がスレーブ FIFO 動作用に切り替わることで FIFO インターフェースを実現しているので，CPU 外部バスに接続した I/O のアクセスとスレーブ FIFO の動作が交錯したり，DMA 転送のときのように外部回路が CPU によるバスアクセスの動きとスレーブ FIFO の転送動作を区別することなく，単に外付けの FIFO メモリがそこにあるかのように，直接データのやりとりが行えるようになっています．

3 GPIF

3.1 GPIF の搭載とスレーブ FIFO の関係

● FIFO を制御するには親が必要

エンドポイントバッファがスレーブ FIFO になるというしかけは，データ転送速度の飛躍的な向上に結び付きました．しかしながら，スレーブ FIFO はあくまでも FIFO であり，外部にプロセッサや FPGA などでインテリジェントな回路を用意し，それらがステータスピンの状態を見ながら FIFO のリード/ライト動作を行わなくてはなりません．つまり，外部回路がスレーブ FIFO の“親”となってデータ伝送の面倒をみなくてはならないわけです．

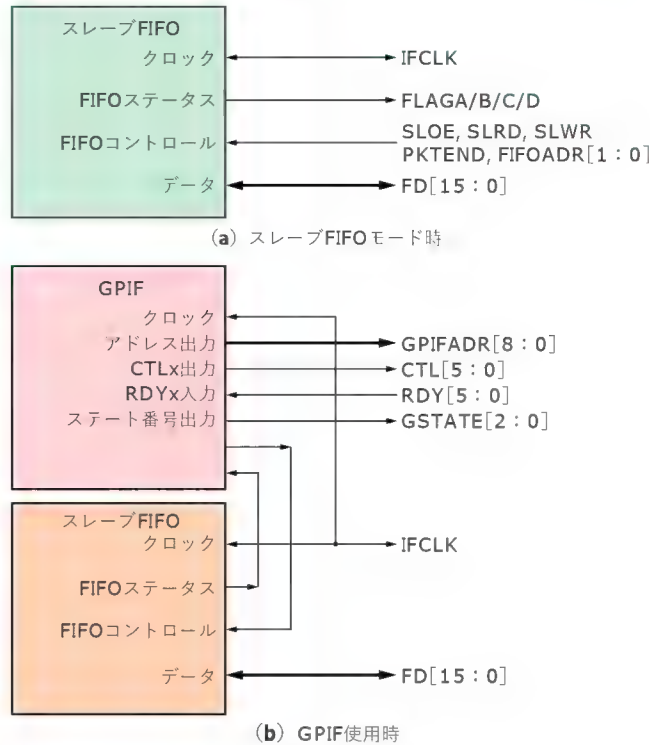
しかしながら，実際のアプリケーションでは，FX2 に接続される相手は必ずしもこのように積極的にデータを取りに来るものばかりではありません．とくに既存の PC 用周辺機器などは，ホスト側からのアクセスを待つようなインターフェースになっているもののほうが圧倒的に多いでしょう．

たとえば，USB2.0 の主要なアプリケーションの一つである USB ハードディスクにしても，IDE ハードディスクは，ホストからのコマンドを受け取って動作するようになっており，データ伝送もホスト側が親となって動作することが前提になっています．

FX2 を使って USB-IDE/ATAPI 変換アダプタを作って USB 2.0 対応 HDD を実現したいとしても，もし FX2 がスレーブ FIFO 機能しかもっていなければ，データ伝送は CPU が 1 回ずつ行うよりほかはなく，スレーブ FIFO が存在する意味がまったくなくなってしまいます．

〔図 8〕スレーブ FIFO と GPIF

(a) スレーブFIFOモード時

(b) GPIF使用時

このような問題に対処するために導入されているのが GPIF です．GPIF は外部の CPU やインテリジェント回路になり代わってスレーブ FIFO の制御を行います．スレーブ FIFO にとっては，GPIF が FIFO 動作の親となり，外部回路にとっては GPIF がホスト側であるかのようにふるまうことで，両者の間のデータ転送を取り仕切るというわけです．GPIF を使わない場合と使う場合のスレーブ FIFO まわりの接続関係をごく簡単に示したのが**図 8** です．

● スレーブ FIFO と GPIF

図 8 からもわかるとおり，GPIF がないときには，外部に直接引き出される FIFO 制御信号は GPIF が処理するかたちになります．GPIF はスレーブ FIFO の親になって，周辺回路とのデータの転送を実行します．親になるという表現だけ聞くと，あたかも DMA コントローラの一種のように思われるかもしれませんが，GPIF は DMA コントローラとはまったく違うコンセプトで設計されています．通常，DMA は外部からの DMA 転送要求などを受けると共通バスのアクセス権を握り，あらかじめ決められたタイミングで，アドレスや DMA アクノリッジ信号，リード/ライト信号などを発生させます．周辺回路を設計するときに，このタイミング図をにらみながら，データの授受ができるようなタイミング生成回路に頭をひねった方も少なくはないでしょう．

これに対して，GPIF はプログラマブルな波形発生機と考えることができます．GPIF の実体は 8 ステートのステートマシンであり，FIFO のステータスや外部からの入力ピンなどは GPIF にとっては特別な意味をもつものではなく，あくまでもステート

マシンのための入力信号という扱いにすぎません．同様に外部出力ピンやスレーブFIFOへの新規のデータラッチ指示や，データの取り出しの指示信号などもまた特殊なものではなく，単なるステートマシンからの出力信号という扱いです．

● GPIFによりプロトコル変換回路が容易に

このように各種の入出力信号をステートマシンの入出力として扱うことにより，General Programmable Interfaceの名称どおり，外部回路アクセスのバスタイミングを自由に設計/変更することができるようになったというのが，GPIFの最大の特徴です．

次章で解説するSRAM接続やFX2間のハンドシェイク接続もそうですが，チップベンダのサンプルにあるUSB-IDE/ATAPI変換アダプタも外付け回路なしで実現できています．また，筆者もFX2を使ったUSB-SCSI変換アダプタを設計したことがありますが，ハード的にはGPIFのデータバスにPLDによるバッファとパリティジェネレータを外付けしただけで，SCSIのハンドシェイクなどはすべてGPIFのステートマシンを使うことで実現できました．

これら以外にも，FX2を既存のマイコンボードとインターフェースするということもいくつか行っていますが，いずれも直結，あるいはごく単純なバッファを接続する程度ですんでしまいました．

3.2 GPIF利用によるデータ転送モード

● GPIFとCPU

ここまでの説明では，GPIFはあくまでもスレーブFIFOの親の役目という位置づけにしていましたが，じつはFX2では，GPIFをFIFOと組み合わせるだけではなく，GPIFとCPUという組み合わせでの利用も可能にしています．つまり，CPUが専用のI/Oポートをリード/ライトすることによって，GPIFを動作させ，スレーブFIFO用のデータバスを使ってデータ転送を行うということもできるのです．

この動作モードのとき，スレーブFIFO用として用意されているFX2のデータバス(FD0～FD15)はCPUがデータリード/ライトするI/Oポートと接続され，CPUからのデータポートアクセスが行われるたびにGPIFが1サイクル(ステート0から始まってステート7に戻るまで)実行して停止するというしくみです．

内部CPUバスのデータバスと外部データバスが直結されるわけではなく，あくまでもレジスタ経由でアクセスするかたちなので，ライト時にはレジスタにデータを書き込んだ時点でCPUは次の命令に移り，これと並行してGPIFによるライト動作が行われます．リード時は，GPIFがリード動作をしてデータがレジスタにラッチされ，これをCPUがリードすることになります．

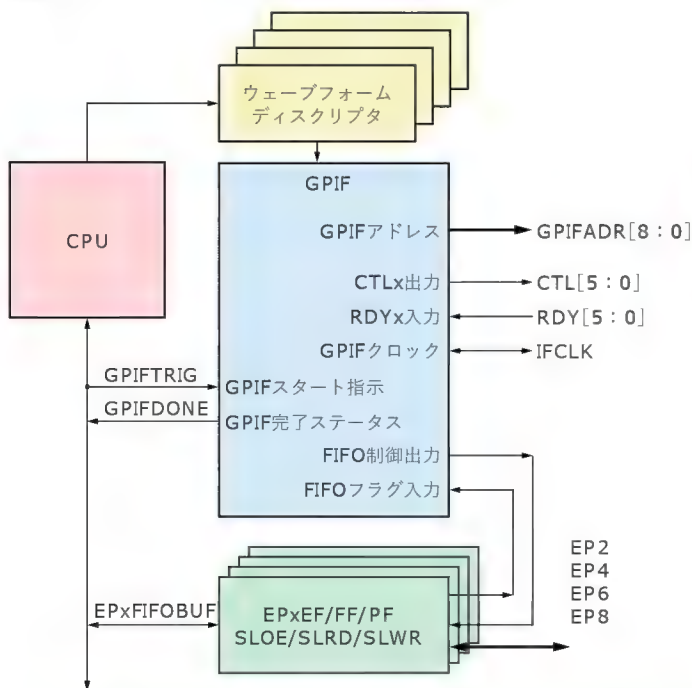
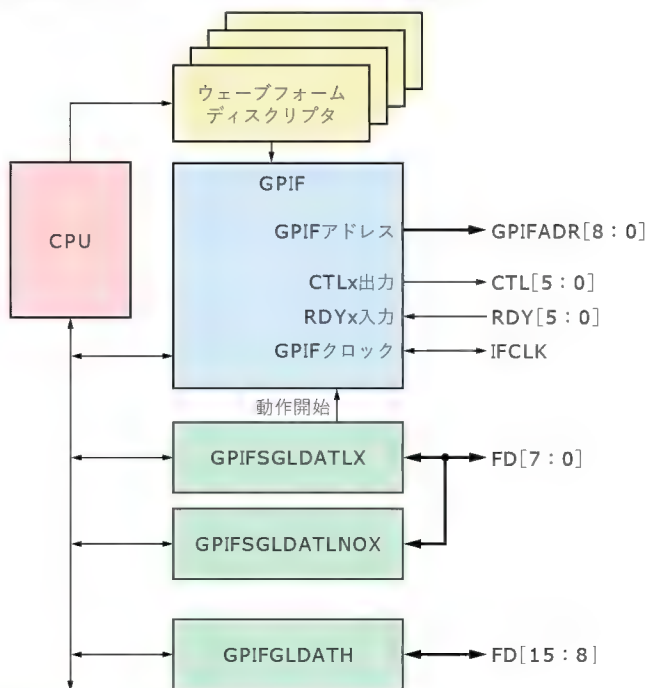
なお，GPIFがアイドル状態にあるときに各ピンの状態を設定するレジスタがあるので，これを利用すれば，通常のPIOポートを1ビットずつ操作して動作させるのと同じような使い方をすることも不可能ではありません．

● FX2のGPIFの利用例

整理すると，FX2のスレーブFIFO用バスの利用方法は，

(a) 外部回路がスレーブFIFOの親になって能動的にリード/ライトする

(b) GPIFがスレーブFIFOの親になって動作し，データはスレーブFIFOとやりとりする

〔図9〕バーストリード/ライト時のGPIF概要

〔図10〕シングルリード/ライト時のGPIF概要

(c) CPUが1回ずつアクセス．GPIFによってリード/ライト動作を行う

という3通りが考えられるということになります．これらのうち(a)はスレーブFIFOモード，(b)がFIFOリード/ライトモード，(c)はシングルリード/ライトモードと名前が付けられています．

なお，マニュアルでは(b)はFIFO-READ/FIFO-WRITEという名称になっているのですが，スレーブFIFOモードと混乱しやすいと思われるので，ここではバーストリード/ライトモードと呼ぶことにしました．

バーストリード/ライト動作時のGPIFやスレーブFIFO関連のブロック接続イメージは，**図9**のようになります．GPIFはウェーブフォームディスクリプタと呼ばれる波形定義テーブルを読み出しながら，エンドポイントバッファ兼スレーブFIFOや外部バス，CPUなどに対してコントロールやステータスの取り込み，状態通知などを行います．

シングルリード/ライト時のイメージは**図10**のように，CPUからアクセス可能なデータレジスタがFIFOバスと接続されます．これらのレジスタへのアクセスを受けてGPIFが外部バスサイクルを生成し，レジスタと外部バスの間でデータ転送が行われます．

● ウェーブフォームディスクリプタ

GPIFの動作を記述するテーブルのことをウェーブフォームディスクリプタと呼んでいます．GPIFが積極的に利用される動作モードは，(b)のバーストリード/バーストライト，および(c)のシングルリード/シングルライトの計四つあるので，動作モードの切り替えのたびにウェーブフォームディスクリプタの再ロードをしなくてもよいように，ウェーブフォームディスクリプタテーブルの領域も4組分用意されています．

後で詳しく説明しますが，4組あるGPIFを4種類のうちいずれの動作モードで動かすかということはCPUによるレジスタアクセス方法によって決定され，実際にどの波形テーブルを使うかということはウェーブフォーム選択用のレジスタ(GPIFWFSELECT)によって決定されます．

たとえば，チップベンダのサンプルであるUSB-IDE/ATAPI変換アダプタでは，IDEデバイスのステータスを読み出したり，コマンドを送出するときにはCPUによる1バイトずつのリード/ライト動作を行うシングルライト/シングルリードが利用され，ディスクのデータの読み出しやデータの書き込みはデータポートを16ビット幅でアクセスし，1セクタ(通常512バイト)単位で一度に転送する，バーストリード/バーストライトを利用しています．

ウェーブフォームディスクリプタが四つ分あることから，初期化のときに各動作モード用のウェーブフォームディスクリプタテーブルをセットしておけば動作中の書き換えは不要であるというわけです．

● プログラマブルステートマシン＝GPIF

シングルリード/シングルライトは，データがスレーブFIFO側のデータバスに出るということ，そしてリード/ライトアクセスの波形をGPIFがコントロールするといった違いはありますが，通常のCPUによる外部バスアクセスと同じように利用することができます．すでに触れたとおりGPIFはステートマシンであり，出力ピンの動作タイミングや，入力ピンをどのタイミングでサンプリングして，それによって次の動作をどうするかということをプログラミングすることができるので，リード/ライト信号などのストローブ信号のタイミングや幅，相手からのREADY信号をどこでサンプリングし，解除された後はどの程度のホールド時間を確保するかということも外部にグルーロジックを設けることなく，すべてGPIFのプログラミング変更だけで対処できてしまうというのは，CPUバスにはない便利な機能です．FX2の中で外部にCPUバスが引き出されているのは128ピンパッケージのものだけですが，GPIFをうまく利用することで，単なる高速伝送用だけでなくフレキシブルな外部バスとして利用することもできるでしょう．

3.3 GPIFの概要

次に，GPIFブロックの構造について見ていくことにします．FX2のGPIFは大きく分けて二つの機能をもっているといえます．一つは，FX2と外部のI/Oデバイスとの間を取り持つ汎用8/16ビットのI/Oインターフェースとしての機能，そしてもう一つは，FX2内部のUSBのエンドポイントFIFOとの連携による高速データ転送処理を行う機能です．まず，バスインターフェースにステートマシンを使うというのはどういうことなのかということについて説明した後，汎用インターフェースとしてのGPIFの機能，USBエンドポイントFIFOとの連携動作について順に説明していくことにします．

● GPIFの構成

FX2のGPIFのブロック図を**図11**に示します．GPIFの要となっているのは8ステート(うち1ステートはアイドルステートなので任意に使えるのは7ステート分)のステートマシンです．ステートマシンとは何かということについては後述しますが，GPIFの機能を簡単にいえばCPUによるPIO転送のときのような，データの更新，外部ステータスのチェックとそれにともなう分岐，外部コントロール信号の制御などといったステップをハードウェアで実現するためのしくみです．

FX2のGPIFの入力には，CPUによるトリガ(GPIFの動作開始ポートアクセス信号)や，外部のRDY*n*ピン信号，内部のUSBエンドポイントFIFOのFULL/EMPTYといったフラグ，クロック数カウンタなどがあります．

また，出力としてはアドレスピンのインクリメント指定やUSBエンドポイントFIFOから次のデータを出力させるための指示信号発生，外部のCTL*n*端子の状態設定などの機能があります．

● ステートマシン

GPIFの動作の要となるのが8ステートのステートマシンです．ステートマシンというのは，現在の状態(ステート番号)を数値

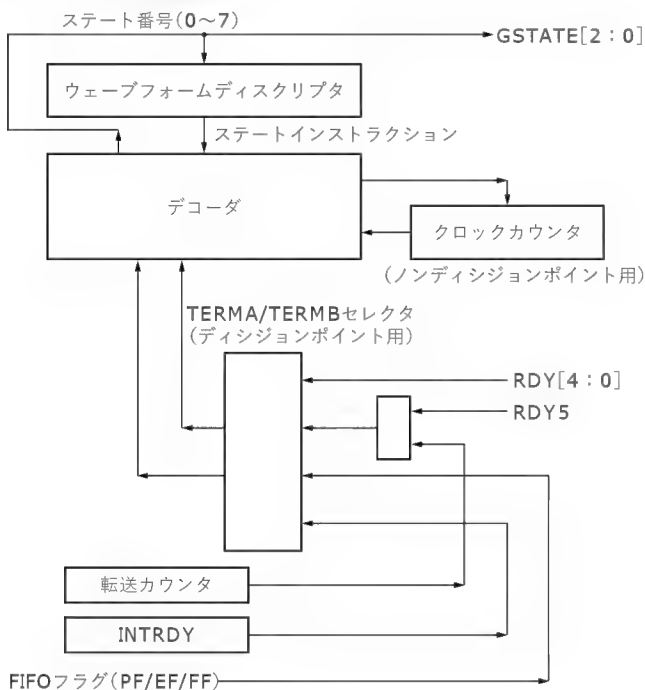
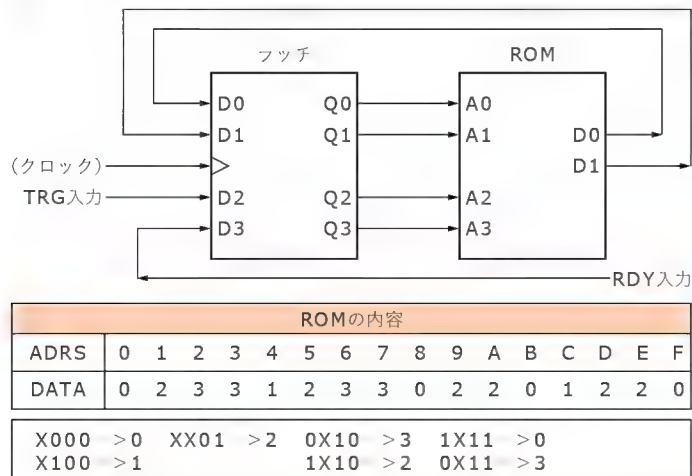
としてもっており，ステート番号に応じて出力を変化させ，またステート番号と入力信号の条件によって次にどのステートに移動するかを決定するというものです．あたかもCPUが条件判定をしながらそれに応じた出力をするのと同じように入力信号の状態によって次の動作が決まり，ステート番号の変化に応じた出力が得られるわけです．単純に入力信号の論理演算だけで出力が得られるのではなく，今の自分の状態と入力信号の状態によって次の動作が決まるような回路が必要になる場面は数多くありますが，このようなところでは必ずステートマシンが利用されていると考えてよいでしょう．

ステートマシンの考え方そのものは，それほど難しいものではありません．**図12**はごく簡単なステートマシンの一例です．

〔図11〕GPIFの構成概略

〔図12〕ステートマシンの例

ROMの内容																
ADRS	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DATA	0	2	3	3	1	2	3	3	0	2	2	0	1	2	2	0

X000 ->0　XX01 ->2　0X10 ->3　1X11 ->0
X100 ->1　　　　　　1X10 ->2　0X11 ->3

ROMには，現在のステート値と外部入力をアドレスとして与えるようにしておき，ROMの中には，与えられたアドレスに対応した次のステート値をデータとして書き込んでおきます．出力をそのまま入力に戻すと発振したようになってしまい，さっぱり安定しなくなるので，クロックを使ってラッチしてからアドレスピンに与えるようにしておきます．このようにしておくと，クロックを与えるたびに入力条件に対応した新しいステート値がROMから出てくるため，あたかもプログラムが実行されているときのように現在の状態と入力条件に対応しながら動作する回路になるわけです．実際のステートマシンでROMチップを使うのは遅すぎるので，この部分が組み合わせ論理回路に置き変わったり，ステート番号の与え方にも高速化のための工夫がなされたりしますが，原理的には同じものと思っておいてかまいません．

● ステートマシンの動作例

さて，ここに掲げたステートマシンの動作を見ていくことにしましょう．**図12**にはROMの中のデータと，データを作る上の考えを表にして示したので，参考にしてください．たとえば，X000 -> 0と書いてあるのは，D1とD0の2ビット(現在のステート値)が0のときにTRG入力(D2)が0であれば，D3(RDY入力)の状態に関係なく次のステート値(D[1：0])は0となることを示しています．これにより，ROMのアドレス0とアドレス8のデータは0となるわけです．

ここで例として取り上げたステートマシンは，四つのステート値をとります．それぞれのステート値のときの動作は次のようになります．手作業で動きを追ってみるとわかりやすいでしょう．

● ステート0

CPUがデータを書き込み，トリガ入力(TRG)が来たらステート1へ(来なければ0のまま)

● ステート1

無条件にステート2へ

● ステート2

RDY入力が'0'ならステート3へ('1'ならステート2のまま)

● ステート3

RDY入力が'1'ならステート0へ('1'ならステート3のまま)

C言語風に書くなら，**リスト1**のようになるでしょうか．

このように，現在のステート値と外部の条件によって次の動作が決定できるというのがステートマシンの特徴です．この例ではステート数は四つですし，入力は全部一度に受けるようになっていますが，FX2のGPIFでは8個のステートがあったり，

〔リスト1〕ステートマシン制御のCプログラム例

```
char STATE=0;
extern bit  RDY;
while(1) {
  switch(STATE) {
     case 0: if (TRG == 1) STATE=1; break;
     case 1: STATE=2; break;
     case 2: if (RDY == 0) STATE=3; break;
     case 3: if (RDY == 1) STATE=0; break;
  }
}
```

同一ステートに一定時間(クロック数で指定)留まる機能や，入力信号の選択とその信号間で論理演算を行い，その結果に応じて分岐先を指定するようになっているなど，マイコン的な色あいが出ているといえるかもしれません．

● コントロール出力

これでステートマシンとしては動作をしましたが，単にステート値がグルグルと変化するだけで何の出力もないということでは，回路としてまったく意味がありません．各ステート値のときに出力ピンの状態をどうするかを決められるようにしておくことで，外部回路に対してストローブ信号やクロックを与えることができるようになります．

FX2のGPIFではCTLn(n：0～5)というピンがコントロール出力信号として用意されており，各ステートにおいてこれらのピンの状態をどうするのかを自由に設定できるようになっています．

たとえば先ほどの例で，ステート1と2のときだけCTL出力ピンの状態が‘1’になり，それ以外のときは‘0’になるようにし，データピンは常にドライブされるようにしておいたとします．CTLをWrite要求信号(‘1’でライト)，RDYを相手のReadyステータス信号(‘1’でレディ，‘0’でビジー)としてタイミングを考えると，

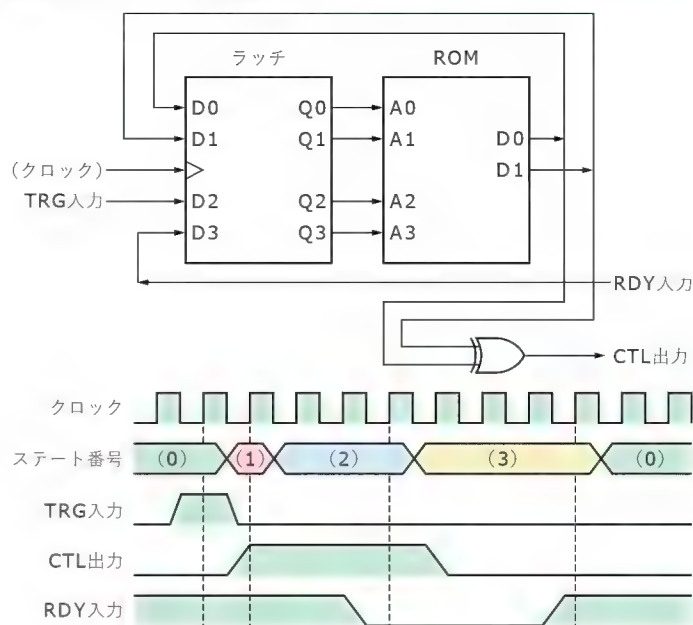
- ステート0
  CPUが書き込み動作を行うまで待つ．書き込まれるとデータバスにデータが現れる
- ステート1
  CTL出力が‘1’になる
- ステート2
  RDY入力が‘0’になるのを待つ
- ステート3
  CTL出力を‘0’に戻し，RDY入力が‘1’になったらステート0に戻る

という具合に動く回路として見ることができるようになります．実際にこのようなコントロール出力が出るようにXORゲートを付加して，タイミング図を書き加えたのが**図13**です．FX2の場合にはこのような相手とのハンドシェイクだけではなく，単にあるクロック分の時間が経過するまでそのステートにとどまるようにすることもできるため，たとえばデータをセットした後，2クロック待ってからCTLを‘1’にして，その後，さらに3クロック待ってからRDY入力をチェックするといったようなことも簡単に行えるようになっています．

また，FX2のGPIFには専用のアドレスピンがあり，この値のインクリメントを指示することも可能なので，外部メモリの連続領域にデータを転送することも簡単です．

● 16ビットシングルライトの手順

スレーブFIFOのデータ入出力バス幅は16ビット幅ですが，8ビット幅としても使用することができます．8ビット幅で使えば1バイトずつ，16ビット幅にすれば2バイトずつ自動的に入出力が行われます．

〔図13〕ステートマシン動作例

シングルリード/ライトのときも，データバスは8ビット幅/16ビット幅のいずれでも使用することができるようになっていますが，FX2内蔵のCPUである8051は古典的な8ビットCPUであり，16ビット単位でのデータ入出力命令はないので，16ビット幅のデータ入出力は上位バイト，下位バイトに分けてアクセスすることになります．

FX2では上位バイト用のポートと，2種類の下位バイトポートが用意されています．一つはアクセスによってGPIFが自動的に起動されるポート，もう一つは単にデータがアクセスできるだけのポートです．

16ビット幅のシングルライト時には，

(1) 上位バイトを書き込む
(2) 自動起動される下位バイトポートに書き込む

という手順を踏むことで，(2)のデータセットとともにGPIFが起動します．GPIFはこのデータポートへのライト動作が行われたことを，シングルライト動作用のウェーブフォームディスクリプタ(GPIFWFSELECTレジスタでどの領域のディスクリプタを実行するかを決める)の実行開始命令と解釈し，動作を開始します．

シングルライト動作を図示したのが**図14**です．8ビット幅のときには(1)のステップが不要です．

● 16ビットシングルリードの手順

一方リード方向の場合は，GPIFを動作させ終わった後でないとデータが読めないので手間がかかります．動作は**図15**にも示すように，次のような手順になります．

(1) 自動起動される下位バイトポートをダミーリードしてGPIFを起動

〔図14〕16ビット幅シングルライト手順

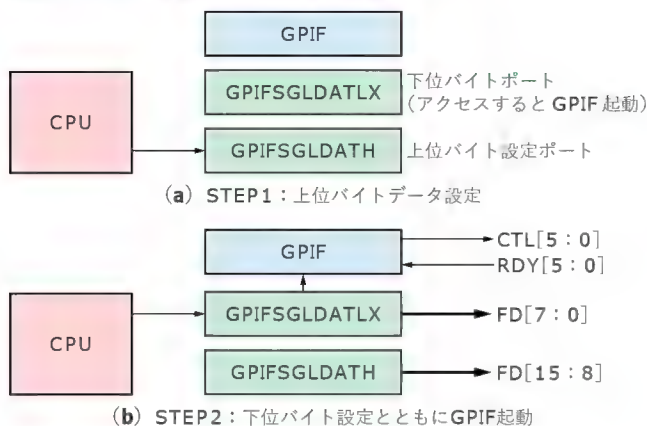
（a）STEP1：上位バイトデータ設定

（b）STEP2：下位バイト設定とともにGPIF起動

〔図15〕16ビットシングルリード手順

（a）STEP1：ダミーリードでGPIF起動＆データ取り込み

（b）STEP2：上位バイト読み出し

（c）STEP3：下位バイト読み出し

(2) GPIFの動作完了待ち
(3) 上位バイトポートを読む
(4) 自動起動しない下位バイトポートを読む

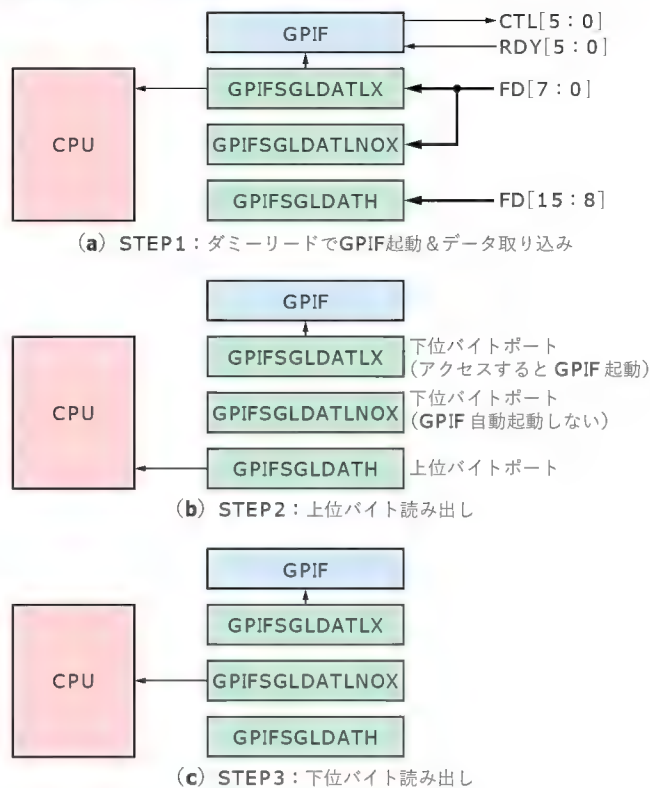
(1)のステップのダミーリードはGPIFに対してシングルリード用のウェーブフォームディスクリプタの内容を実行させる起動命令に相当します．起動した後，CPUはGPIFの動作完了を待って(4)のステップで自動起動しないほうのデータアクセスだけを行うポートを読み出せば，データが取り込まれて終了となるわけです．

もし，(4)に続いてすぐ次のデータを読み出すならば，(4)のステップで自動起動する側のポートを読み出すようにすれば，2ワード目からのリードは(2)のステップから行えばよいことになります．このような使い方をすると，CPUが読み出したデータの処理をするのと並行してGPIFが動いて，次のデータリード動作を行うので効率の良いデータ転送が行えます．

ライト方向のときに自動起動しないポートに書き込むと，そのデータはそのまま出力データとしてラッチされ，データ出力がイネーブルになっていれば（GPIFIDLECSレジスタを使用する）そのまま通常のPIOポートのように出力されます．コントロール信号もGPIFIDLECTLレジスタを使えば任意の状態に設定できるので，無理矢理CPUでバスサイクルを作ることも不可能ではありませんが，そのような使い方をすることはほとんどないでしょう．

3.4　GPIFと関係信号

GPIFの内部構造がわかってきたところで，次にGPIFと外部のインターフェース信号について見ていくことにします．FX2のGPIFと外部のインターフェース信号には，次のようなものがあります．

- IFCLK（GPIFの動作クロック入出力）
- CTL0～CTL5（出力端子）
- RDY0～RDY5（入力端子）
- GPIFADR0～GPIFADR8（アドレス出力）
- GSTATE0～GSTATE2（現在のステート値出力）
- FD0～FD15（データバス）

すでに説明したとおり，FD0～FD15はGPIFではなく，スレーブFIFOのほうに分類されるものですが，一体として動作するものなので，ここではGPIFに含めておきました．次に，各信号について少し説明を補足しておきます．

● IFCLK（GPIF動作クロック入出力）

IFCLK端子はGPIFの動作クロック入出力端子で，**図16**に示すような構成になっています．図中，IFCFG.6などとなっているのはFX2内部のIFCFGレジスタのビット6を示します．

図16からもわかるように，GPIFの動作クロックとしてはFX2内部で生成される48MHzや30MHzの内部クロック，あるいは外部クロックのいずれかから選択可能です．IFCLKピンは内部クロック使用時にはクロック出力として，外部クロックとして使うときにはクロック入力として利用できるようになっています．

IFCFG.4はクロックの反転機能です．‘1’にすることで，内部クロックと外部クロックの位相を反転させることができるようになっています．GPIFのステートマシンは内部クロックの立ち上がりエッジに同期して動きますが，外部回路では反転したほうが都合がよい場合もあります．たとえば，外部回路側もクロックの立ち上がりエッジに同期して動く場合，クロックとデータの位相関係が問題になってきます．安全を考えるなら，上側のタイミング図のように，最初のクロックの立ち上がりエッジ

でデータ更新，次の立ち上がりエッジでデータ取り込みという具合に交互に行うことになりますが，反転クロックを利用すれば，FX2と外部回路の両者が半クロックずれて動作するため，毎クロックエッジごとにデータが更新できます．このようなときに外部にゲートを入れて反転するのではなく，FX2のレジスタ操作のみで反転できるようにしているのです．

IFCFG.5は，内部クロックをIFCLK端子からクロックを出力するか否かを決めるものです．'1'のときはIFCLKが出力になり，'0'だと出力されません．IFCLKを入力ピンとして使うときは'0'のままにしておきます．

IFCFG.6は，動作クロックとして内部クロックを利用する場合の周波数選択ビットです．'0'になっていると30MHzの内部クロックが，'1'になっていると48MHzの内部クロックが選ばれます．

IFCFG.7は，GPIFのクロック源として内部クロックを使うか，外部クロックを使うかを選択するビットです．'1'のとき内部クロック，'0'のときは外部クロック(IFCLKからの入力)が使用されます．

● CTL0～CTL5(コントロール出力)

GPIFからの出力信号です．56ピンパッケージのFX2では出力されているのはCTL0～CTL2のみですが100ピン，128ピンのFX2はCTL0～CTL5まですべて利用可能です．

CTL出力は大きく二つのグループに分かれています．ここでは，CTL0～CTL3の4ビットを仮に下位グループ，CTL4とCTL5を上位グループと呼ぶことにします．

▶コントロール出力ピンのモード設定

GPIFのコントロール出力のうち，下位グループのほうは通常のトーテムポール出力("H"か"L"をドライブ)かオープンドレイン出力として使うか，あるいは3ステート出力として使うかを設定可能です．上位グループのほうはトーテムポール出力かオープンドレインのいずれかです．

この設定には複数のレジスタが絡んでおり少々ややこしいのですが，まとめると**表1**のようになります．表はGPIFがアイドル(非動作)状態のときを示したため，GPIFIDLECTLレジスタが使用されるように記載していますが，GPIF動作中はGPIFIDLECTLレジスタの代わりにGPIFのウェーブフォームディスクリプタ中のステートインストラクション(後で説明する)のOUTPUTフィールドの値が使用されます．

CTL出力をどのモードにするかはGPIFCTLCFGレジスタによって決まります．GPIFCTLCFG.7(TRICTLビット)が，下位グループをトライステートとして使うか否かの設定ビットで'1'ならばトライステートモードになり，上位グループが無効になります．

〔図16〕IFCLK系統図と反転クロックの効果

GPIFCTLCFG.7(TRICTL)ビットが'0'に設定され，トーテムポール/オープンドレインモードになったときは下位グループ，上位グループの両方とも各ビットごとにトーテムポールとするか，オープンドレインとするのかを指定することができます．

GPIFCTLCFG.0～GPIFCTLCFG.5がそれぞれCTL0～CTL5をトーテムポールにするのか，あるいはオープンドレインにするのかの設定ビットになります．該当するビットが'0'ならばトーテムポール出力，'1'ならばオープンドレイン出力になります．

一方，GPIFCTLCFG.7を'1'に設定し，3ステートモードにしたときには他のビットの設定に関係なく，下位グループは3ステートモード，上位グループは無効となり，使用できなくなります．

〔表1〕CTL出力モード設定

	GPIFCTLCFG.7(TRICTL)			
	0		1	
	GPIFCTLCFG[5：0]		GPIFIDLECTL[7：4](注)	
	0(トーテムポール出力)	1(オープンドレイン出力)	0	1
CTL[5：4]	GPIFIDLECTL[5：4](注)の値		無効	無効
CTL[3：0]	GPIFIDLECTL[3：0](注)の値		ハイインピーダンス	GPIFIDLECTL[3：0]の値

注：GPIF動作中はステートインストラクションのOUTPUTフィールドの値

▶コントロール出力ピンの出力設定

コントロール出力状態の設定は，GPIF が非動作状態のときやGPIF のステートマシンがアイドルステート（ステート7）にあるときにはGPIFIDLECTL レジスタの設定値によって決まり，動作中（ステート0～ステート6）にいるときには，後で説明する，GPIF のウェーブフォームテーブルの中の OUTPUT フィールドの値で決定されます．どちらもフォーマットは同じです．上位グループ側のビットの意味が出力モードによって変わるところに注意してください．

GPIFCTLCFG.7（TRICTL ビット）が'1'のときは上位4ビットはコントロール出力の下位グループの3ステート制御用のビットとなり，'0'ならばハイインピーダンス状態になりますが，このとき，GPIFIDLECTL[7：4]（GPIF 動作中はステートインストラクションの OUTPUT フィールドのビット7～4）がそれぞれCTL[3：0]出力の3ステート制御ピンになります．GPIFIDLECTL[7：4]の該当するビットが'1'ならば出力がイネーブルになりGPIFIDLECTL[3：0]の設定に応じた出力となり，'0'ならばハイインピーダンス状態になります．

たとえば，GPIFIDLECTL が3Ah ならば，

CTL0 ："H"出力
CTL1 ："L"出力
CTL2 ：ハイインピーダンス（チップ内部では'1'をラッチ）
CTL3 ：ハイインピーダンス（チップ内部では'0'をラッチ）

となります．

〔図17〕SAS ビットの働き

〔図18〕GPIFADRH/GPIFADRL レジスタ

● RDY0～RDY5（レディ入力）

RDY は名称からするとレディ信号のように見えますが，実際にはGPIF のステートマシンへの入力信号であるというだけで，使用方法は任意です．RDY 入力ピンは56 ピンパッケージのFX2ではRDY0 とRDY1 だけが使用可能です．100 ピン，128 ピンのものではRDY0～RDY5 まですべて使用することができます．

RDY0～RDY5 の入力は内部でGPIF 動作クロックを使ってラッチされてから使用されます．RDY 入力への入力信号がGPIF の動作クロックと同期しているときには1段ラッチでよいのですが，GPIF 動作クロックと非同期に動いている場合，クロックと入力の変化タイミングによっては正しくデータをラッチできない（メタステーブルをおこす）可能性があるので，FX2 では非同期モードのときに対応してRDY 入力を2段ラッチ（1回内部クロックでラッチした出力をもう一度ラッチする）することができるようになっています．

理屈からわかるとおり，2段ラッチの場合，1段ラッチの場合よりも1クロック分GPIF の応答が遅くなります．RDY 入力を2段ラッチにするか否かを決めるのが，GPIFREADYCFG レジスタ（アドレス：E6F3h）のビット6（SAS ビット）で，'0'のときは1段ラッチ，'1'のときは2段ラッチになります．これを図示したのが**図17**です．

ただ，実際に波形を見るかぎりでは，マニュアルのこの説明とは逆に，SAS が'1'のときが1段，'0'のときが2段ラッチになったような動きを示します．今回実験したGPIF に FX2 同士の非同期データ転送でも，SAS ビットを'1'にした方が応答が早くなる一方，伝送速度を上げていくと途中で異常動作を起こすという，1段ラッチのときのような動きを示すことから，SAS ビットは'0'に設定しました．

● GPIFADR0～GPIFADR8（アドレス出力）

GPIF からの9ビットのアドレス出力です．56 ピンパッケージ品にはなく，100 ピン，128 ピンパッケージ品のみ出力されています．**図18** はGPIFADRH/GPIFADRL レジスタとINCAD，GPIFADR 出力の関係を図示したものです．

FX2 の外部にメモリなどをつないで，パケットデータを転送するような場合，連続したアドレス領域にアクセスしたいということがよくあります．このような目的のために，GPIF では9ビットのアドレスをもち，初期値の設定，およびウェーブフォームテーブルに設定することで，自動的にアドレスをインクリメントすることができるようにしています．

GPIFADR は，汎用I/O ポートのうちPORTE のビット7（PORTE.7），およびPORTC の全ビット（PORTC[7：0]）がそれぞれGPIFADR8，GPIFADR[7：0]として切り替えられるようになっていて，アドレス出力として使用しない場合には汎用I/O ポートとしても使用可能です．汎用ポートとして使用するか，GPIFADR として使用するのかはPORTECFG やPORTCCFG

によってビットごとに設定可能となっています．

GPIFADRピンとして設定した場合の初期値はGPIFADRL(アドレス：E6C5h)，GPIFADRH(アドレス：E6C4h)によって，任意の値に設定可能です．また，GPIFのステートマシンによる自動インクリメントの指示はウェーブフォームテーブルの中のINCADビットで行われます．ウェーブフォームディスクリプタ中にインクリメント指示がなければ，アドレス出力は設定された値のまま保持されるので，ある固定されたアドレスへの連続アクセスにしたり，各ビットをチップセレクト端子のように使うことも可能です．

● GSTATE0～GSTATE2(ステート値出力)

IFCONFIG(アドレス：E601h)のビット2(GSTATE)を‘1’にすると，汎用I/OピンのうちPORTE[2：0](PORTE.2～PORTE.0の意)ピンが現在のGPIFのステート値を示す出力ピンになります．

GPIFのステートマシンは，CPUから独立して動作するうえ，シングルステップ実行のようなことが行えません．ロジックアナライザで波形を見ながらウェーブフォームテーブルのデバッグをするときに便利なように，PORTEの下位3ビットに現在のステート値を出力することができるようになっているわけです．このステート値を外部でデコードするなどしてウェーブフォームテーブルだけでは作りきれないような波形を生成することも可能です．

● FD0～FD15(データバス)

何度か触れてきたように，データバスはスレーブFIFOやCPU用のI/Oポートなので，厳密にはGPIFには含まれませんが，一体となって動作するものであることから，GPIFのデータバスという扱いにして説明しておきます．

データバスが入力として使われるか，出力として使われるかは，GPIFにトリガをかけるときに決まります．GPIFにシングルライトやバーストライトの動作開始指示をした場合には出力に，シングルリード，バーストリードなら入力になります．したがって，GPIFを使ってデータを出力した後に読み出すといったようなことを，1回のGPIF動作によって(一つのウェーブフォームテーブルによって)行うことはできません．

このような場合には，CPUによってライト方向の動作をスタートさせ，それが完了したあと，今度はリード方向の動作を行わせるという手順を踏む必要があります．

GPIFのデータバスは8ビットバス，あるいは16ビットバスのいずれでも使用可能で，どちらで使うかはEP*n*FIFOCFG(*n*は2，4，6，8のいずれか)レジスタのビット0(WORDWIDEビット)で決定されます．EP*n*FIFOCFGのWORDWIDEビットがどれか一つでも‘1’になっていれば，GPIFのデータバスは16ビット幅になります．すべて0ならばデータバスは8ビット幅となり，上位8ビットは汎用I/Oポート(PORTD)として利用可能です．第2章のサンプルではメモリアクセスを8ビット幅で，FX2間のデータ転送は16ビット幅で行いました．

3.5 GPIF関連レジスタ

FX2の内部レジスタはオリジナルの8051との互換性を保ちながら，かなり欲張った拡張を施していることから，レジスタの構成はかなり複雑になっています．とくにGPIF関連のレジスタは，同じような機能を果たすために複数のレジスタが設けられているなど，必ずしも整理されているとは言い難いものがあること，マニュアルの説明もあちこちに分散していて，全体像が非常につかみにくいものになっており，混乱しやすいのではないかと思われます．

各機能についての詳細はマニュアルを読んでいただくことにし，ここではGPIF，USBエンドポイントバッファ/スレーブFIFOの動作に関係するレジスタのうち主要なものについて**表2**(pp.62-64)にまとめておいたので参考にしてください．

● ウェーブフォームディスクリプタ

GPIFの動作を決定するのがウェーブフォームディスクリプタです．なかなかいかめしい名称ですが，実体はデータテーブル用のメモリであり，書き込まれるデータはCPUでいうところのプログラムに相当するものです．GPIFのステートマシンはステート0からステート7までの8ステートの状態をとるので，いわば8ステップのプログラムが組めるようになっていると見ることができます．

アイドル状態，すなわち非動作状態のとき動作モードに関係なく，GPIFはステート7になっています．このことからステート7はアイドルステートとも呼ばれます．アイドルステートにある状態からGPIFに起動がかかると，GPIFは，シングルリード/ライト，バーストリード/ライトのウェーブフォームディスクリプタの中から起動された方法に対応したテーブルのステート0から実行を始めます．たとえば，GPIFSGLDATLXレジスタへの書き込み動作であったら，シングルライト動作が開始されるわけです．起動されたGPIFはステート0からスタートしてステート7で終了します．

これを状態遷移図で表すと，**図19**(p.65)のようになります．

ウェーブフォームディスクリプタはGPIFが動作していないときであればいつでも書き換え可能です．たとえばIDE/ATAPIインターフェースでも，もっともクラシックなモード0から始まってUDMAモードにいたるまで，さまざまな転送モードがありますが，このような場合，まずモード0に対応したデータをウェーブフォームテーブルにセットし，その後は対応可能な最高速モードにセットし直すといったこともできるわけです．

実際に，チップベンダの評価ボードによるUSB-IDE/ATAPI変換アダプタサンプルでも，このようなウェーブフォームディスクリプタの書き換えを行っています．

● ウェーブフォームディスクリプタの構造

すでに触れたとおり，FX2では4組分のウェーブフォームディスクリプタ領域が確保されています．GPIFは8ステートのステートマシンなので，1組のウェーブフォームディスクリプタも

〔表2〕FX2のレジスター覧

名　称	領　域	リード/ライト	アドレス	ビット7	ビット6	ビット5	ビット4	ビット3	ビット2	ビット1	ビット0	設定内容
WAVEDATA	XDATA	R/W										ウェーブフォームメモリ #0(デフォルトではFIFOリード用)
			0xE400	LENGTH/BRANCH[0]								
				Number of IFCLK cycles to stay in this state(0 = 256cycles)								DPビットが'0'のとき，このステートに留まるクロック数(IFCLK)を指定0は256クロックの意味になる
				Re-Execute	x	BRANCHON1			BRANCHON0			DPビットが'1'のとき次に移動するステートや再実行の指示 Re-Execute ='1'：同一ステートに飛んだときINCADやNEXTを再実行する　'0'：同一ステートのときは再実行しない BRANCH1：LOGIC FUNCTIONの論理演算の結果が'1'のとき分岐する先のステート番号 BRANCH0：LOGIC FUNCTIONの論理演算の結果が'0'のとき分岐する先のステート番号
			0xE401	LENGTH/BRANCH[1]								
			0xE402	LENGTH/BRANCH[2]								
			0xE403	LENGTH/BRANCH[3]								
			0xE404	LENGTH/BRANCH[4]								
			0xE405	LENGTH/BRANCH[5]								
			0xE406	LENGTH/BRANCH[6]								
			0xE407	予約								
			0xE408	OPCODE[0]								
				x	x	SGL	GINT	INCAD	NEXT/SGLCRC	DATA	DP	このステートにおける動作を指定 SGL ='1'：バーストモードのとき，強制的にシングル転送用レジスタを使う．シングル転送時は無効 GINT ='1'：GPIFWF割り込みを発生させる(INT4に入ってくる) INCAD ='1'：GPIFのアドレスバス(GPIFDR)の値をインクリメントする NEXT/SGLCRC：SGLビットによって機能が変わる ●SGL ='1'：NEXT/SGLCRC ='1'ならUDMA = CRCH/Lレジスタ　'0'ならS GLDATAH/Lレジスタ ●SGL ='0'：NEXT/SGLCRC ='1'ならFIFOポインタを進める．　'0'なら現状維持 DATA： ●DATA ='1'：リード時はデータバス(FD0～FD15)上のデータをラッチ　ライト時はデータバスをドライブ ●DATA ='0'：リード時はデータバス(FD0～FD15)上のデータは無視　ライト時はデータバスをハイインピーダンスにする DP ='1'：ディシジョンポイント(入力条件判定して分岐) '0'：ノンディシジョンポイント(指定クロック数留まって次に進む)
			0xE409	OPCODE[1]								
			0xE40A	OPCODE[2]								
			0xE40B	OPCODE[3]								
			0xE40C	OPCODE[4]								
			0xE40D	OPCODE[5]								
			0xE40E	OPCODE[6]								
			0xE40F	予約								

(a) GPIFウェーブフォームディスクリプタ

〔表2〕FX2 のレジスタ一覧(つづき)

名称	領域	リード/ライト	アドレス	ビット7	ビット6	ビット5	ビット4	ビット3	ビット2	ビット1	ビット0	設定内容
WAVEDATA	XDATA	R/W	0xE410	OUTPUT[0]								
				OE3	OE2	OE1	OE0	CTL3	CTL2	CTL1	CTL0	GPIFCTLCFG の TRICTL ビットが'1'のときのビット配置： OE＝'1'CTL*n* 出力イネーブル　'0'：ハイインピーダンス
				x	x	CTL5	CTL4	CTL3	CTL2	CTL1	CTL0	GPIFCTLCFG の TRICTL ビットが'0'のとき
			0xE411	OUTPUT[1]								
			0xE412	OUTPUT[2]								
			0xE413	OUTPUT[3]								
			0xE414	OUTPUT[4]								
			0xE415	OUTPUT[5]								
			0xE416	OUTPUT[6]								
			0xE417	予約								
			0xE418	LOGIC FUNCTIN[0]								
				LFUNC		TERMA			TERMB			入力の選択と，論理演算の指定 LFUNC：論理演算の指定 00：A　AND　B 01：A　OR B 10：A　XOR　B 11：NOT(A)　AND　B TERMA/TERMB：入力指定 000：RDY0 001：RDY1 010：RDY2 011：RDY3 100：RDY4 101：RDY5/転送カウント終了(READYCFG.5＝'1'のとき) 110：スレーブ FIFO のフラグ(ProgrammableFlag，EmptyFlag，FullFlag) 111：INTRDY(内部レディ)GPIFREADYCFG.7 の値
			0xE419	LOGIC FUNCTIN[1]								
			0xE41A	LOGIC FUNCTIN[2]								
			0xE41B	LOGIC FUNCTIN[3]								
			0xE41C	LOGIC FUNCTIN[4]								
			0xE41D	LOGIC FUNCTIN[5]								
			0xE41E	LOGIC FUNCTIN[6]								
			0xE41F	予約								
		R/W	0xE420～0xE43F	0xE400～0xE41F と同様								ウェーブフォームメモリ #0(デフォルトでは FIFO ライト用)
		R/W	0xE440～0xE45F	0xE400～0xE41F と同様								ウェーブフォームメモリ #0(デフォルトではシングルリード用)
		R/W	0xE460～0xE47F	0xE400～0xE41F と同様								ウェーブフォームメモリ #0(デフォルトではシングルライト用)

(a) GPIF ウェーブフォームディスクリプタ(つづき)

〔表2〕FX2のレジスター覧(つづき)

名称	領域	リード/ライト	アドレス	ビット7	ビット6	ビット5	ビット4	ビット3	ビット2	ビット1	ビット0	設定内容
GPIFCTLCFG	XDATA	R/W	0xE6C3	TRICTL	0	CTL5	CTL4	CTL3	CTL2	CTL1	CTL0	GPIFのコントロール出力の動作モード設定 TRICTL=‘1’：トライステートモード．CTL0～CTL3のみ有効 ‘0’：トーテムポール/オープンドレイン出力．CTL0～CTL5有効 CTLx=‘1’：オープンドレイン出力 ‘0’：トーテムポール出力

(b) GPIFコントロール出力設定

名称	領域	リード/ライト	アドレス	ビット7	ビット6	ビット5	ビット4	ビット3	ビット2	ビット1	ビット0	設定内容
GPIFIDLECTL	XDATA	R/W	0xE6C2	CTLOE3	CTLOE2	CTLOE1	CLTOE0	CTL3	CTL2	CTL1	CTL0	TRICTL=‘1’のときのビット配置 ● CTLOEx=‘1’：CTLx出力イネーブル ‘0’：CTLx出力はハイインピーダンス ● CTLx=‘1’：出力は“H” ‘0’：出力は“L”(いずれもCTLOE=‘1’のとき)
				0	0	CTL5	CTL4	CTL3	CTL2	CTL1	CTL0	TRICTL=‘0’のときのビット配置 ● CTLx=‘1’：出力は“H”またはハイインピーダンス(オープンドレイン時)．‘0’：出力は“L”

(c) GPIFアイドル状態出力

名称	領域	リード/ライト	アドレス	ビット7	ビット6	ビット5	ビット4	ビット3	ビット2	ビット1	ビット0	設定内容
GPIFREADY CFG	XDATA	R/W	0xE6F3	INTRDY	SAS	TCXRDY5	0	0	0	0	0	READY入力の設定 INTRDY：GPIFのステートインストラクションがディシジョンポイントのときの，TERMA/TERMB入力として利用 SAS=‘1’：RDY入力は内部クロックと非同期(内部クロックで2段ラッチする) ‘0’：RDY入力は内部クロックと同期(1段ラッチ) TCXRDY5=‘1’：GPIFのディシジョンポイントステートインストラクションでRDY5入力を転送カウンタのカウント終了フラグと入れ換え

(d) GPIFのRDY入力設定

8ステート分の領域から構成されています．また，1ステート分の状態や条件判断などの記述は「ステートインストラクション」と呼ばれており，一つのステートインストラクションは4バイトのデータから構成されています．つまり，1組のウェーブフォームディスクリプタは32バイト〔＝4(バイト/ステート)×8(ステート)〕で構成されているわけです．

ウェーブフォームディスクリプタとステートインストラクションの関係を示したのが**図20**です．たとえばステート0の記述には0xE400，0xE408，0xE410，0xE418の4バイトが使われ，ステート3ならば0xE403，0xE40B，0xE413，0xE41Bの4バイトによって記述されます．ウェーブフォームディスクリプタの並びは動作モードごとに並んでいますが，ステートインストラクションの並びは各フィールドごとにステート0から7まで並ぶという形式になっていることに注意してください．

なお，1組のウェーブフォームディスクリプタには8ステート分のステートインストラクションを記述できますが，このうちユーザーが自由に設定できるのはステート0～6の七つです．ステート7はアイドルステートといって，GPIFが非動作状態にあるときのデフォルトの位置として使われるので，ステート7用に相当するデータ領域に書かれたステートインストラクションは無視されます．

● ディシジョンポイントとノンディシジョンポイント

GPIFを使って外部とインターフェースする場合，当然相手の回路とタイミングをあわせなくてはなりません．一般的な手法は，次のようなものでしょう．

(1) メモリアクセスのように，あらかじめ決められたタイミングにしたがってパルス幅などを確保する

(2) 相手からのREADY信号などによってウェイトしたり，ハンドシェイクを行う

これらのいずれか一方だけという場合もあれば，ハンドシェイクする場合でもセットアップタイムやホールドタイムを確保するために両方を組みあわせて利用すること

も少なくありません．FX2 の GPIF もこれに対応して，あるクロック数分そのステートに留まるようにするのか，あるいは入力信号の状態によって次のステート番号を決めるのかをステートインストラクションで選択することができるようになっています．

FX2 のマニュアルの中では，前者のような一定クロック数待つステートをノンディシジョンポイント，後者のように入力信号の状態で分岐するものをディシジョンポイントと呼んでいます．あるステートのステートインストラクションがディシジョンポイントであるのか，ノンディシジョンポイントであるのかは，ステートインストラクションの DP ビット（0xE408～0xE40E の OPCODE フィールドのビット 0）によって決定されます．このビットが‘1’ならばディシジョンポイント，‘0’ならばノンディシジョンポイントになります．

たとえば，WRITE 信号をアサートして 2 クロック待ってから相手からの READY 信号がアサートされるのを待つということならば，ノンディシジョンポイントで WRITE 信号をアサートしたまま 2 クロック待たせ，その次のステートはディシジョンポイントにして，READY 信号がアサートされていたら次のステートへ，されていなければ同一のステートに留まるようにすれば良いわけです．

図 21 にディシジョンポイントとノンディシジョンポイントの両方を使った例を示します．ステート 3 から 7 への無条件分岐（BASIC や C の goto に相当）はディシジョンポイントで演算結果が‘0’でも‘1’でも同じステートに飛ぶようにすることで実現します．

● ノンディシジョンポイントのステートインストラクション

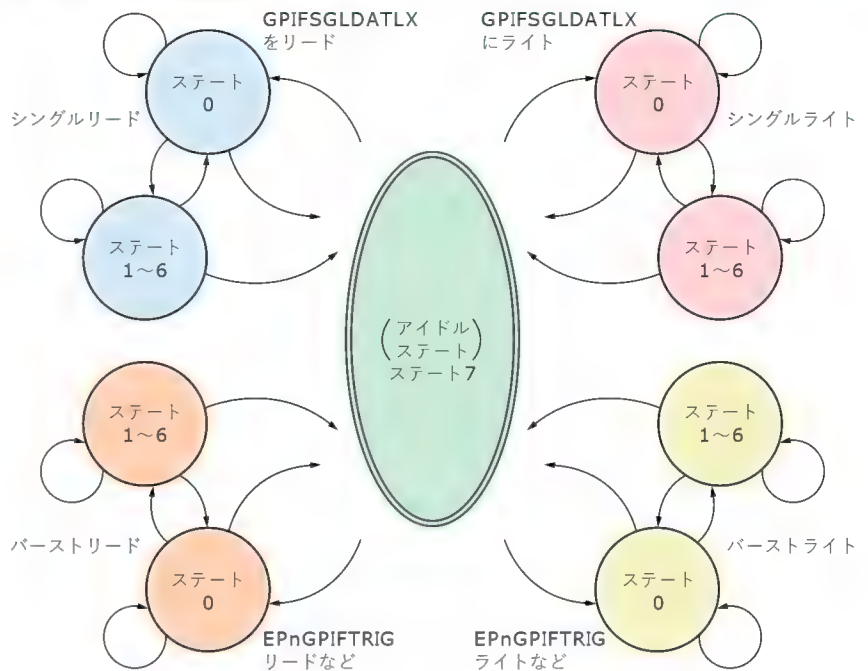
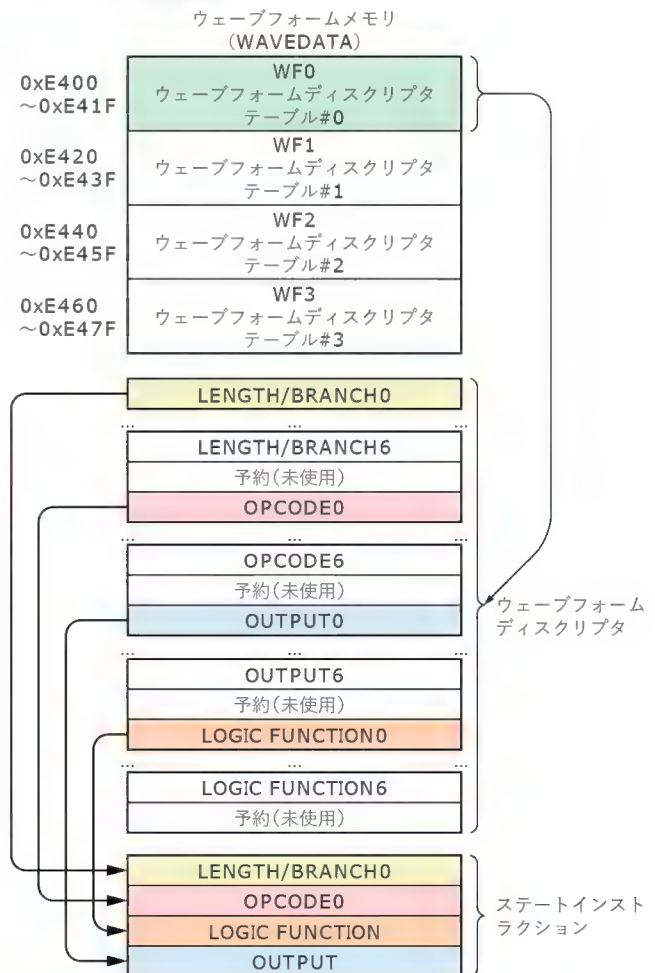
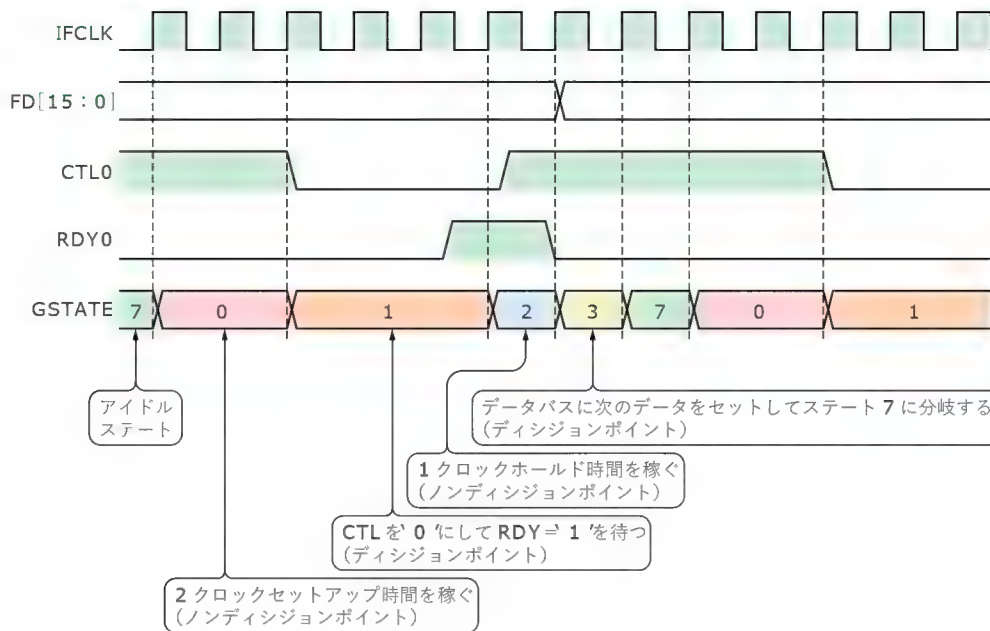
ノンディシジョンポイントのステートインストラクションは**表 2** のレジスタマップのウェーブフォームディスクリプタの説明のうち「DP =‘0’のとき」のようになります．ステートインストラクションは，

- LENGTH/BRANCH
- OPCODE
- LOGIC FUNCTION
- OUTPUT

の 4 バイトで構成されますが，このうち LOGIC FUNCTION はノンディシジョンポイントでは使用されません．

▶ LENGTH/BRANCH

LENGTH/BRANCH バイトはこのステートに留まる時間を GPIF クロックのクロック数で指定するものです．最小は 1 クロックで，0 にすると 256 クロックの意味になります．GPIF のクロックに 48MHz の内部クロックを利用しているならば，1 クロ

〔図 19〕GPIF の状態遷移

〔図 20〕ウェーブフォームディスクリプタとステートインストラクションの関係

〔図 21〕
ディシジョンポイントとノンディシジョンポイントの使い分け

ックあたり約20.8nsですから，最長約5.3μsまで留まらせることができます．

▶ OPCODE

OPCODEバイトは，このステートにおける内部動作の指示を行うもので，各ビットは次のような機能となっています．

● DP

このステートインストラクションがディシジョンポイントであるか(DP＝‘1’)，ノンディシジョンポイントであるのか(DP＝‘0’)選択するものです．今説明しているのはノンディシジョンポイントなのでDP＝‘0’です．

● DATA

スレーブFIFOへのデータ入出力の指示ビットです．WRITE方向(FX2から外部への向き)のときはDATAが‘1’だとデータバスがドライブされ，‘0’だとデータバスはハイインピーダンス状態になります．READ方向のときには，DATAが‘1’だとデータバス上のデータがスレーブFIFOにラッチされます．

少し注意が必要なのは，バーストリードとライトのときではFIFOのポインタ(CPUからは見えない)の進み方が違うということです．WRITE方向のときはDATAビットはデータバスのゲートを開く信号という扱いなので，‘1’であっても，FIFOのポインタは動きません．DATAが‘1’のステートがいくつあっても同じデータが出続けます．FIFOのポインタを進めて次のデータを出力するようにするには，次に説明するNEXTビットを使います．

一方，READ方向のときはDATAビットが‘1’になっているとデータがFIFOにラッチされ，ポインタが進みます．このため，通常READ方向のウェーブフォームディスクリプタの中でDATAビットが‘1’になっているステートは一つだけです．

● NEXT/SGLCRC

このビットは，バーストリード/ライト動作のときだけ意味をもちます．また，このビットはビット5のSGLビットによって意味が変わってきます．SGLビットが‘0’のときはこのビットはNEXTビットになり，ライト方向の動作のときにFIFOのポインタを進めてFIFOの中の次のデータを出すかどうかの選択になります．NEXTビット＝‘1’ならば次のデータが出力されるようになり，‘0’ならばそのままです．

ライト方向の動作のときには，スレーブFIFOへの切り替えが行われた段階で，FIFOのポインタはすでにFIFOの先頭をさしているため，DATAビットを‘1’にするとFIFOの先頭データが出力されます．このため書き込み動作を開始する前のステートでNEXTを‘1’にすると，先頭データが捨てられてしまうことになります．バーストライトのときには書き込み完了のステートのステートインストラクションにおいてNEXT＝‘1’にして次のデータが出力されるようにするのが一般的な使い方でしょう．

SGLビットが‘1’のとき，NEXT/SGLCRCビットはSGLCRCの意味になります．SGLビットが‘1’のときには強制的にシングルリード/ライトになるのですが，このときデータバスと接続されるレジスタがSGLDATAH/SGLDATALか，UDMA_CRCH/UDMA_CRCLのいずれにするかを切り替えるのがSGLCRCビットです．このビットが‘1’ならばUDMA_CRCH/UDMA_CRCLの値が，‘0’ならばSGLDATAH/SGLDATALの値が用いられます．

● INCAD

GPIFは9ビットのアドレス出力をもっており，初期値をCPUからGPIFADRH，GPIFADRLレジスタによって設定できるよ

うになっています．INCADビットはこのアドレス出力の値をインクリメントするものです．単に出力アドレスがインクリメントされるだけであって，スレーブFIFOのポインタなどにはいっさい影響ありません．

● GINT

FX2の内蔵CPU，8051への割り込み要求を行うビットです．このビットが1になっているステートインストラクションをGPIFが実行したとき，CPUに対してGPIFWF割り込み（8051のINT4入力に割り付けられている）が発生します．

● SGL

バーストリード/ライトモードのときだけ意味をもつビットです．通常，バーストリード/ライトモードのときには外部バスはスレーブFIFOと接続されますが，このビットが‘1’になっていると，データバスはスレーブFIFOではなく，NEXT/SGLCRCで選ばれたレジスタになります．

この機能が実装されることになった背景は，FX2でUSB-IDE/ATAPI変換アダプタを実現したいということにあったようです．UDMAモードではデータの信頼性を上げるために生データの後にエラー検出用のCRCデータが付加することになっています．ところがUSBのマスストレージクラスでは，コマンドやデータに対してCRCデータを付加してくれません．このため，ライト時のCRCデータの送出や，リード時のCRCデータの受け取り/チェックなどはFX2側で処理しなくてはなりません．リード方向のCRCデータはエラーチェックをしないことにして捨ててしまうという方法もありますが，ライト方向のCRCデータはそういうわけにはいきません．この対策として，CRCデータのみシングル転送モードに切り替える機能を追加したということのようです．

▶ OUTPUT

OUTPUTバイトはGPIFが管理している6本のコントロール出力ピン（CTL0～CTL5）の状態設定を行うものです．コントロール出力ピンはトーテムポール出力（“H”か“L”のいずれかのみ出力）か，オープンドレイン出力にするのか，3ステート出力にするかを選択可能です（選択はGPIFCTLFGレジスタと，GPIFCTLCFG.7のTRICTLビットで行う）．

TRICTLビットを‘1’にしてGPIFのコントロール出力を3ステートにしたしたときには上側のようになり，CTL0～CTL3までの4ビットが使用可能となります．上位4ビットのOE0～OE3がアウトプットイネーブル（OE*n*＝‘1’でイネーブル，OE*n*＝‘0’ならハイインピーダンス）となり，下位4ビットのCTL0～CTL3がデータになります．このとき，CTL4，CTL5は使用不可となります．

一方，TRICTLビットを‘0’にしたときには下側のようなビット配置になり，6ビットすべてが利用できるようになります．出力がトーテムポールになるのか，オープンドレインになるのかはGPIFCTLCFG[5：0]によって個別に指定できます（ステートインストラクションの中で変更することはできない）．

● ディシジョンポイントのステートインストラクション

ディシジョンポイントのステートインストラクションの場合は，LENGTH/BRANCHバイトがBRANCHとして機能すること，LOGIC FUNCTIONバイトが意味をもってくるところがノンディシジョンポイントとは違ってきます．OUPUTやOPCODEフィールドの内容はノンディシジョンポイントと同じです．

▶ LENGTH/BRANCH

LENGTH/BRANCHフィールドは，このステートの次に実行するステートや実行方法を指定するものです．

● BRANCHON0/BRANCHON1

後で説明するLOGIC FUNCTIONの結果が‘1’ならば，BRANCHON1の3ビットで指定されるステートへ，‘0’ならばBRANCHON0で指定されるステートに分岐します．分岐先には現在のステートを指定してもかまいません．たとえばBRANCHON0に今のステート番号を入れれば，LOGIC FUNCTIONの結果が‘1’になるまで（つまり演算結果が‘0’である間はずっと）現在のステートに留まるという動きになります．

なお，ステート7はアイドルステートなので，最後にステート7に分岐させるか，あるいは，ステート6がノンディシジョンポイントで，LENGTHで指定したクロックだけ経過してステート7に移行した段階で1回の転送動作が完了します．バーストリード/ライトの場合に終了条件を満足していなければ，自動的に再度ステート0からステートインストラクションが実行されます．

● Re-Execute

分岐先が現在と同じステートであった場合に，FIFOとのデータのやりとりを現状ホールドにするか，あるいは他のステートから移動してきたときのように実行する（Re-Execute：再実行）かを選択するビットです．

通常，BRANCHONxによる分岐先が現在と同じステートになっていた場合，GPIFは現ステートの状態をホールドします．たとえば，バーストライトのときのステートインストラクションでINCADやNEXTビットが‘1’，BRANCHON0が自分自身のステートをさしているときには次のような動きになります．

(1) 他のステートからこのステートにくると，GPIFのアドレスバスがインクリメント．FIFOから次のデータを出力

(2) BRANCHON1側の条件が成立するまでその状態を維持

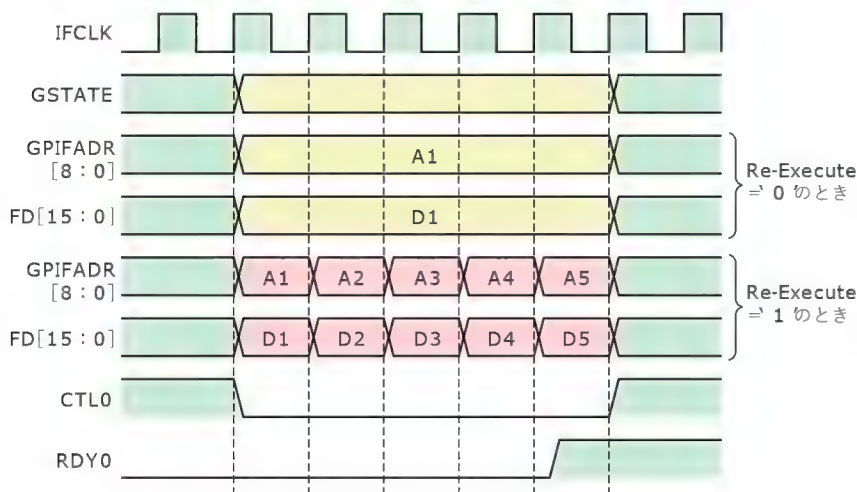
これに対して，ステートインストラクションのRe-Executeビットが‘1’になっていると，

(1) 他のステートからこのステートにくると，GPIFのアドレスバスがインクリメント．FIFOから次のデータを出力

(2) BRANCH1側の条件が成立するまで1クロックごとにアドレスバスをインクリメント，FIFOからも新しいデータを出力

となります．両者の違いを**図22**に示すので参考にしてください．

Re-Executeモードの場合，ステートの移動をともなわずにデータが連続出力されるので，GPIFの転送能力を最大限に引き出すことができるモードであるということができます．外部回路がIFCLKと同期して動いており，しかも毎クロックエッジごとにデ

〔図22〕Re-Execute による動作の違い

ータ転送が可能であるような場合には非常に便利なモードです．

▶ LOGIC FUNCTION

LOGIC FUNCTION バイトは，次のステートを LENGTH/BRANCH バイトの BRANCH0 側にするか，BRANCH1 側にするのかを判定するための入力と論理演算方法を指定するものです．TERMA，TERMB がそれぞれ GPIF に与えられている入力信号選択で，その両者の間で LFUNC で指定されている論理演算が行われます．この結果が‘0’ならば BRANCH0，‘1’ならば BRANCH1 で指定されるステートに分岐します．

GPIF のステートマシンでは，1 ステートについて入力を二つまで指定して，双方の AND，OR，XOR，片方の反転ともう一方の AND のいずれかの演算結果が‘1’であるか‘0であるかによって，次にどのステートに移行するかを設定できるようになっています．入力となりうるのは，RDY0～RDY5，トランザクションカウンタのカウント終了，エンドポイント FIFO のフラグ(Empty，Full，Programmable から選択)，GPIFREADYCFG レジスタ(アドレス：E6F3)のビット 7(INTRDY ビット)で，これらの中から任意の二つを選択できます．

今回のサンプルにあるように，結果が‘0’でも‘1’でも同じステートに行くようにすれば，無条件分岐命令になりますし，判定する入力が一つだけの場合は，同一入力同士の演算にすればよいのです．たとえば TERMA，TERMB とも 000(RDY0)，LOGIC　FUNCTION も“00”(AND)にするといった具合に同じ入力同士で AND や OR 演算すれば，RDY0 の状態によって分岐させることができます．

三つ以上の入力によって判定しなくてはならない場合には，ステートを複数個使って判定することになります．たとえば，RDY[2：0]が“001”という状態になったら転送動作を開始するような場合には，

● ステート 0

RDY2 と RDY1 の OR をとって，‘1’ならステート 0 へ．‘0’ならばステート 1 へ

● ステート 1

RDY0 同士で AND をとって‘0’ならステート 0 へ．‘1’ならステート 2 へ

● ステート 2～6

データ転送動作用

という具合に割り付ければよいわけです．

GPIF のステートのうち自由に使えるのは 7 ステート(一つはアイドルステート)しかありませんし，判定に 1 クロック分の遅れが生じるので，複数のステートを使用する場合にはステート不足になったり，入力信号の変化タイミングに注意が必要であるということには注意してください．

● **TERMA/TERMB**

TERMA，TERMB は論理演算の入力選択です．“000”～“100”はそれぞれ FX2 の外部端子である RDY0～RDY4 になります．“101”～“111”の三つは少々入り組んでいるので注意が必要です．

TERMA/TERMB =“101”のときの入力は GPIFREADYCFG.5 (GPIFREADYCFG レジスタのビット 5)によって切り替わります．GPIBREADYCFG.5 が‘0’ならば FX2 の RDY5 入力になり，‘1’ならばトランザクションカウンタ(転送カウンタ)のカウント終了(Transaction-Count Expiration)フラグが指定されます．カウントが終了したとき‘1’，それまでは‘0’です．

TERMA/TERMB =“110”のときは，スレーブ FIFO の出力するフラグが入力となります．スレーブ FIFO フラグには PF (Programmable Flag)，EF (Empty Flag)，FF (Full Flag)から選択します．この選択は EPxGPIFFLGSEL (x はエンドポイント番号：2，4，6，8)によって決定されます．

TERMA/TERMB =“111”のときは，INTRDY (Internal Ready)となります．INTRDY は GPIFREADYCFG.7 (GPIFREADYCFG レジスタのビット 7)で，これは CPU によってリード/ライトすることが可能です．これによって，CPU からのレジスタへの設定を待って実行させたり，INTRDY の内容によって処理を振り分けるということが簡単に行えます．

● **LFUNC**

TERMA/TERMB で選ばれた入力同士の間の論理演算を指定します．以前のバージョンでは“00”：AND，“01”：OR，“10”：XOR のみでしたが，現在は“11”：A の反転と B の AND という条件が追加されています．

参考文献

1) TECH I Vol.8，『USB ハード＆ソフト開発のすべて』，CQ 出版(株)
2) 桑野雅彦，「USB2.0 対応 USB 学習キット新登場！」，Interface，2001 年 11 月号

くわの・まさひこ　パステルマジック

FX2を使って10Mバイト/秒を超える転送レートを実現する

高速転送対応USBターゲットの設計事例

桑野雅彦

FX2を活用するうえで重要なスレーブFIFOとGPIFについて解説したところで，本章では具体的な応用事例を解説する．もっとも基本的なデバイスの接続事例としてSRAMを接続し，Windows上から読み書きする例を示す．次に，同じFX2ボード2枚を対向して接続し，片方を送信用に，もう片方を受信用としてデータ通信を行う事例を解説する．

（編集部）

1 メモリデバイスの接続事例

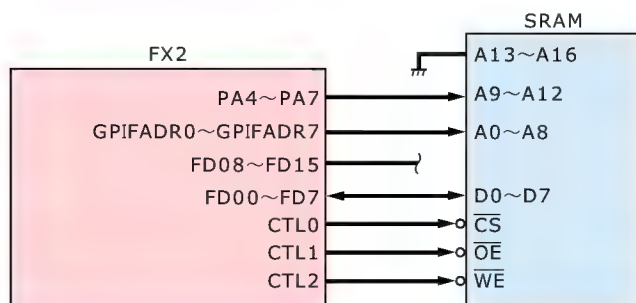
● SRAMの接続

それでは，実際にGPIF（General Pragrammable Interface）を動作させてみることにします．まず簡単な例としてSRAMを接続してみます．使用したSRAMは秋葉原で安価に手に入るということで，HM628128ALFP-10（日立）を使ってみました．

いささか古すぎるデバイスであるためか，日本のサイトからはデータシートはすでに消えてしまっていますが，後継品のHM628128DLP-5のデータシートが米国のサイト（http://semiconductor.hitachi.com/1m.html）からダウンロード可能です．

SRAMとFX2のGPIF信号との接続ブロック図を**図1**に示します．

- GPIFのアドレスバス（GPIFADR[8：0]）とSRAMのアドレスピンの下位8ビット（A[8：0]）
- GPIFのデータバス（FD[7：0]）とSRAMのデータピン
- GPIFのCTL0とSRAMの$\overline{\mathrm{CS1}}$ピン
- GPIFのCTL1とSRAMの$\overline{\mathrm{OE}}$ピン
- GPIFのCTL2とSRAMの$\overline{\mathrm{WE}}$ピン

さらに今回は，SRAMのアドレス4本分（A[12：9]）を汎用ポートのPORTA（PA[7：4]）で補うことで，16バンク×512バイト（計8Kバイト）分のメモリとして利用できるようにしています．あくまでもバンク切り替えのイメージで，バースト転送時の自動桁上がりなどはサポートしていないので，もし必要ならばソフトウェアでアドレスをチェックして適宜バンク切り替えを行うということになります．

今回のファームウェアではこの機能をサポートしていないので，512バイトバウンダリを超える転送は先頭アドレス側にラップラウンドしてしまいます．

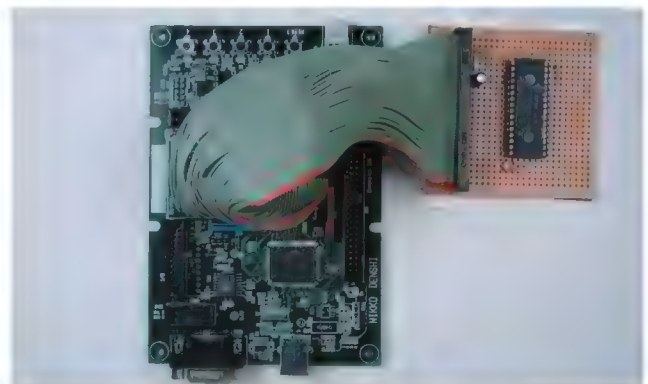
実際の回路図を**図2**（次頁）に示します．HM628128はチップセレクト信号を二つもっていますが，今回は“L”アクティブのほうだけを使うので，“H”アクティブのCS2は単にプルアップしています．

なお，今回使用したSRAMは5V動作のものですが，FX2の入力は5Vトレラントなので，レベルコンバータなどは入れずに簡単に直結で済ませました．試作ボードのようすを**写真1**に示します．

● ソフトウェアインターフェース仕様

今回のファームウェアではバルクOUTエンドポイント，バルクINエンドポイント，そしてコントロールエンドポイント（EP0）

〔図1〕SRAMとの接続ブロック図

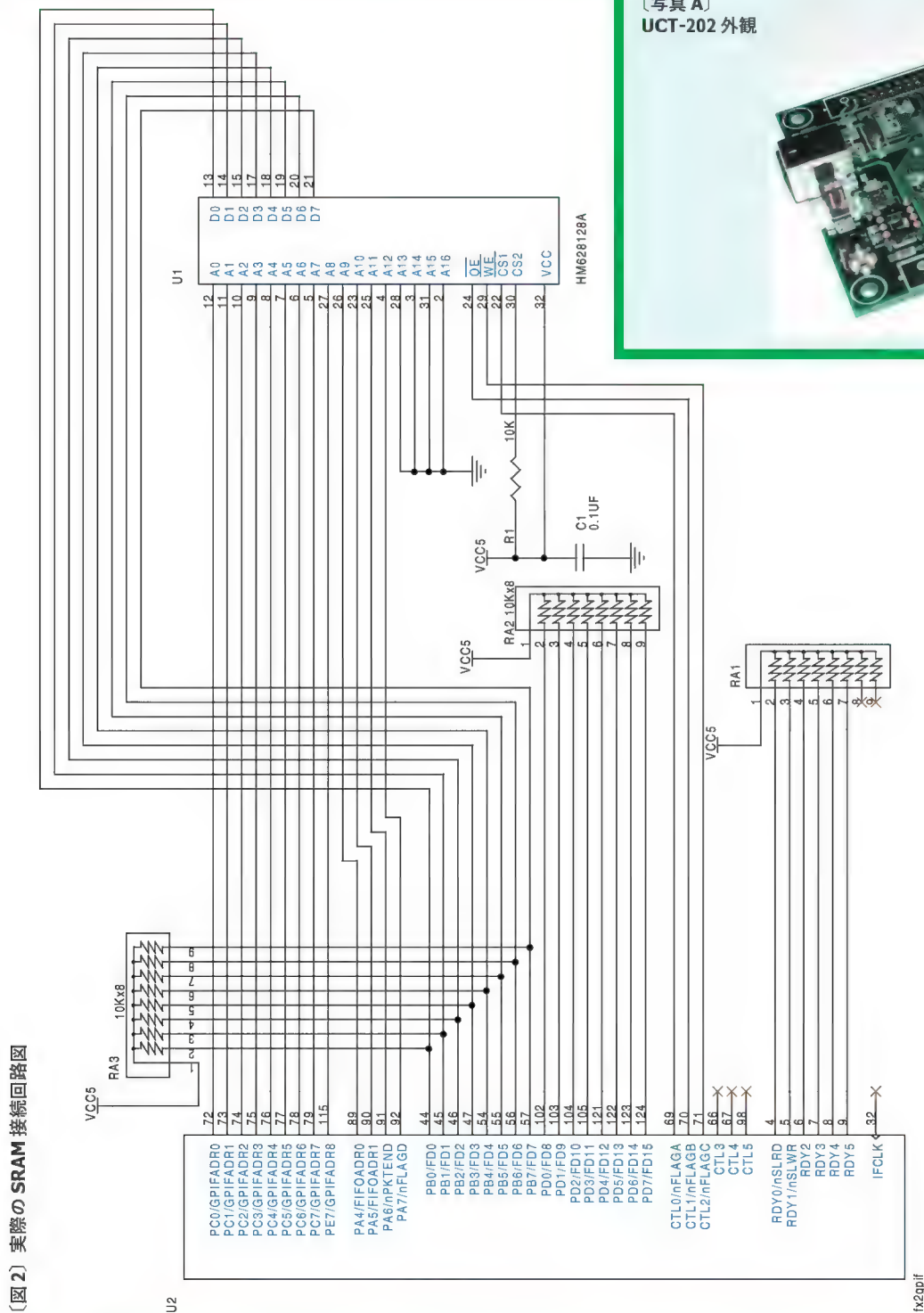
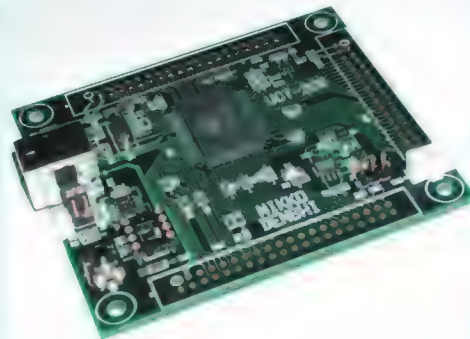
〔写真1〕SRAMを接続したFX2ボード

コラム　EZ-USB FX2 実装済み評価ボード

今回の特集に合わせて，(株)ニッコー電子から，低価格FX2評価ボードUCT-202(**写真A**)が発売されます．128ピンのFX2を採用し，I/Oピンがすべて端子に引き出されているので，デバイスの評価，各種実験や組み込み用途などに利用可能です．

1枚からでも購入可能です．趣味に仕事に，FX2を活用してください．

■ UCT-202 問い合わせ先
(株)ニッコー電子
TEL：03-3625-4668
URL：http://www.nikko-denshi.co.jp/usb/
価格：9,800円

〔写真A〕UCT-202 外観

〔図2〕実際のSRAM接続回路図

を使い，バーストライト，バーストリード，シングルライト，シングルリードの四つの転送モードをサポートします．

FX2は，バルクIN/OUTに使える汎用のダブルバッファのエンドポイントを4組もっていますが，今回は二つしか使わないので，二つのクワッドバッファ(4バンク)のエンドポイントとして使用することにしました．ハイスピードモードで接続されたときには，各エンドポイントのサイズが512バイトになるので，512 × 4 = 2048バイトまでは自動的にバッファリング可能となります．

バーストライト/リード，シングルライト/リードのそれぞれの転送動作は次のように行わせてみることにしました．

▶バーストライト

バルクOUTエンドポイント(EP2)に来たデータをそのままGPIFを使ってバーストライト動作で書き込みます．GPIFADRは1ワードの転送ごとに自動的にインクリメントします．転送先アドレスの初期値はアドレス設定のベンダリクエストによって事前に設定しておきます．

▶バーストリード

ベンダリクエストによって，アドレスの初期値とデータ長を与えると，そのバイト数分のデータをバーストリード動作を行い，転送します．要求サイズがパケットサイズ以上であった場合，パケットサイズ単位への分割はソフトウェアで行います．

▶シングルライト

ベンダリクエストでホストからアドレスとデータを指定します．FX2側では指定されたアドレスをアドレスバス(PA[7：4]，GPIFADR[8：0])に設定し，データを1バイト，シングルライト動作によって書き込みます．

▶シングルリード

ベンダリクエストでアドレスが指定されるので，そのアドレスをアドレスバスにセットし，シングルリード動作を行います．読み出されたデータはEP0を使って送り返します(データINステージをともなうベンダリクエストとして実装)．

● ベンダリクエスト

これらに対応して，**表1**のような次の四つのベンダリクエストを用意しました(バーストライトはリクエスト不要)．

▶アドレスセットアップ要求(bRequest = 0x80)

wValueが設定したいアドレスです．受け取ったアドレスの下位9ビットをGPIFADRH/GPIFADRLに，ビット9～12の4ビットをPAの上位4ビット(PA[7：4])にセットします．データステージはともなわないので，wLengthフィールドは00hです．

▶バーストリード開始要求(bRequest = 0x81)

wValueにリード開始アドレス，wIndexに転送したいデータ長を指定します．ただし，今回のサンプルでは，先に触れたとおりアドレスのインクリメントはGPIFの自動インクリメントに頼っているので，512バイトバウンダリをまたぐリード転送はできません(0番地にラップラウンドする)．データステージはともなわないので，wLengthフィールドは00hです．

▶シングルライト要求(bRequest = 0x82)

wValueにライトしたいアドレス，wIndexの下位8ビットに書き込みたいデータをセットすると，アドレスバス用のレジスタ(IOA[7：4]，GPIFADRH，GPIFADRL)にアドレスを設定した後，データをシングルライト動作によってデータを書き込みます．wIndexの上位8ビットは無視されます．データステージはともなわないので，wLengthフィールドは00hです．

▶シングルリード要求(bRequest = 0x83)

wValueでアドレスを指定すると，アドレスバスにアドレスを設定した後，シングルリードでデータを1バイト読み込み，EP0を使ってデータを送り返します．コントロール伝送のデータINステージを利用するため，wLengthには転送データバイト数(01h)を入れます．

● ファームウェアの作成

FX2の内蔵CPUは，FXシリーズと同じ8051互換コアが採用されています．8051用のCコンパイラはいくつかありますが，今回のファームウェアはサンプルということもあり，フリーのSDCC(Small Device C Compiler：http://sdcc.sourceforge.net/)を用いました．

チップベンダ推奨の商用コンパイラであるKeil社のものなどに比べると，生成されるコードの効率が悪いことや，現バージョンではまだコード生成を間違うパターンがいくつかあるため，バグを回避するような記述が必要になるといった問題はありましたが，それでもFX2を使ったUSB-SCSI変換アダプタのファームウェアは実現できたので，今回のサンプル程度のものに使うには十分でしょう．

● GPIFによるシングルリード/ライト動作の設計

まず，シングルリード/ライト用のウェーブフォームディスクリプタを設計します．今回はGPIFを48MHzの内部クロックで動作させるようにして，**図3**のような波形としてみました．

● ステート0

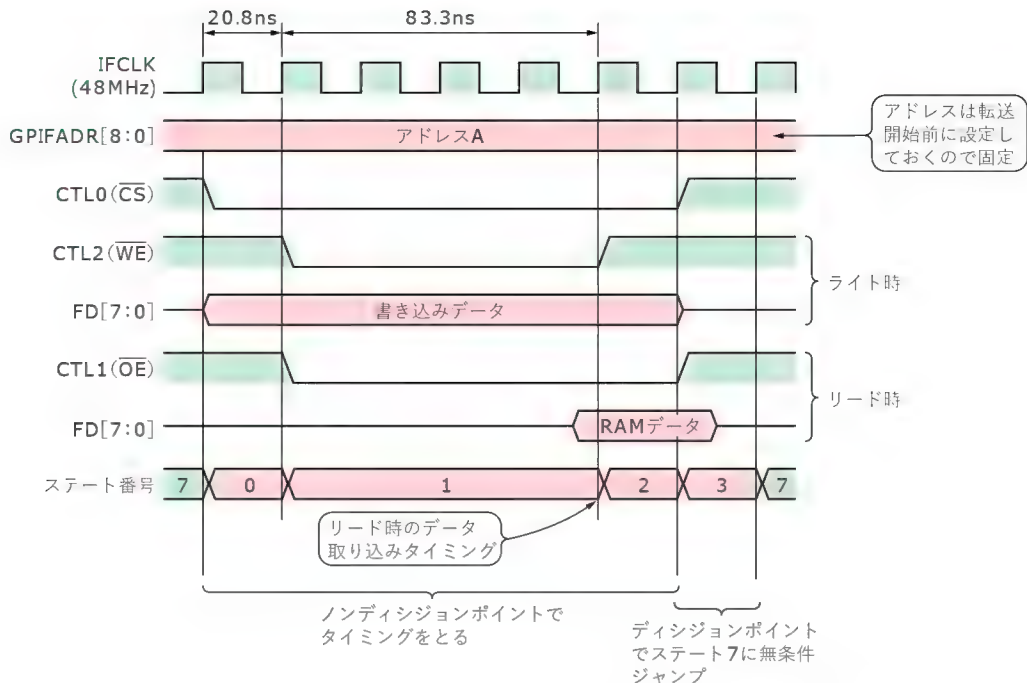
SRAMの$\overline{CS}$をアサート．ライトのときはデータバスドライブ開始

● ステート1

SRAMの$\overline{WE}$(ライト時)または$\overline{OE}$(リード時)をアサートし

〔**表1**〕**FX2-SRAM転送用ベンダリクエスト**

コマンド	bmRequestType	bRequest	wValue	wIndex		wLength	(data)
アドレスセットアップ	0x40	0x80	アドレス	(未使用)		0x0000	なし
バーストリード	0x40	0x81	アドレス	転送データ長		0x0000	なし
シングルライト	0x40	0x82	アドレス	(未使用)	バイトデータ	0x0000	なし
シングルリード	0xc0	0x83	アドレス	(未使用)		0x0001	リードデータ

〔図3〕SRAMシングルアクセスタイミング案

て4クロック(約83nsウェイト)待つ

- ステート2

ライト時は$\overline{\mathrm{WE}}$ネゲート．リード時はデータバス上のデータを取り込む

- ステート3

リード時は$\overline{\mathrm{OE}}$ネゲート．ステート7(アイドルステート)に無条件分岐

また，CTL出力はすべてトーテムポール出力として，アイドルステートではCTL[2:0]はすべて“H”状態にします．

この考えに基づいて作成したシングルライト用のウェーブフォームディスクリプタやレジスタの設定，およびアイドル時用のレジスタ設定をまとめたのが**図4**です．波形とつきあわせて見るとわかりやすいでしょう．ステート3の無条件分岐はRDY0同士のAND条件として，結果が‘1’でも‘0’でもステート7に移行するという記述にして実現しています．

同様にシングルリード時の設計は**図5**のようになります．ステート2でDATA=‘1’となっているので，ここでデータがレジスタに取り込まれます．

● GPIFによるバーストリード/ライトの設計

次に，バーストリード/ライトを考えてみることにします．基本的なアクセス波形はシングルリード/ライトと同じなので，ウェーブフォームディスクリプタも大部分は流用可能ですが，バーストのときには，1回の転送が終わった後で，GPIFADRの値をインクリメントしたり，ライト時にはスレーブFIFOから出力されるデータを次のデータ位置のものにする必要があります．

バーストアクセスのタイミングは**図6**(p.74)のようにしてみました．図に示したように，アドレスの更新，ライト時のデータ更新をステート3で行っています．

これによるウェーブフォームディスクリプタの変更箇所は，**図7**(p.74)のようになります．バーストライトのほうは，ステート3でINCADとNEXTビットをともに‘1’にしてGPIFADRのインクリメントとデータの更新を行います．

バーストリードのときのデータラッチは，シングルリードと同様にDATAビットで行われ，NEXTビットは使用されないので，INCADによるアドレス更新を行うところだけが変わります．

● サンプルファームウェアの基本構造

サンプルファームウェアでは，スタートアップ部分ではリセット後の初期化やUSBの標準リクエストの処理を行い，USBデバイスとして認識されるところまでを標準部分(`fx2sram.rel`：オブジェクトで提供)の中で行い，それ以外の処理はユーザー側に任せるような構造にしました．具体的には標準部分でレジスタの初期化を行った後に，

- ユーザーによるFX2のUSB関連レジスタの初期化〔`usr_reginit()`〕

が呼ばれ，さらにディスクリプタテーブルなどが初期化された後で，

- ユーザーによる初期化〔`usr_init()`〕

が呼び出されます．この後，割り込みが許可となり，メインルーチンである`usr_task()`がコールされます．今回はこの`usr_task()`の中でバーストリード，バーストライト処理を行いました．

また，USB割り込みのうち，GET_DESCRIPTORなどの標準リクエストは`fx2sram.rel`側で処理しますが，ベンダリクエストや，エンドポイント割り込みなどはユーザー側の関数がコールされます．今回はベンダリクエスト以外のものは使っていないので，ベンダリクエストだけコマンドの判定，処理関数

〔図4〕**SRAMシングルライト用ウェーブフォームディスクリプタの設計**

LENGTH=0x01							
0	0	SGL=0	GINT=0	INCAD=0	NEXT=0	DATA=1	DP=0
LOGIC FUNCTION(未使用)							
0	0	CTL5=0	CTL4=0	CTL3=0	CTL2=1	CTL1=1	CTL0=0

(**a**) ステート#0

LENGTH=0x04							
0	0	SGL=0	GINT=0	INCAD=0	NEXT=0	DATA=1	DP=1
LOGIC FUNCTION(未使用)							
0	0	CTL5=0	CTL4=0	CTL3=0	CTL2=0	CTL1=1	CTL0=0

(**b**) ステート#1

LENGTH=0x01							
0	0	SGL=0	GINT=0	INCAD=0	NEXT=0	DATA=0	DP=1
LOGIC FUNCTION(未使用)							
0	0	CTL5=0	CTL4=0	CTL3=0	CTL2=1	CTL1=1	CTL0=0

(**c**) ステート#2

Re-Exec=0	0	BRANCHON1=0x07			BRANCHON0=0x07		
0	0	SGL=0	GINT=0	INCAD=0	NEXT=0	DATA=0	DP=1
LFUNC=00		TERMA=00			TERMB=00		
0	0	CTL5=0	CTL4=0	CTL3=0	CTL2=1	CTL1=1	CTL0=1

(**d**) ステート#3

TRICTL=0	0	CTL5=0	CTL4=0	CTL3=0	CTL2=0	CTL1=0	CTL0=0

全部トーテムポール出力にする

(**e**) GPIFCTLCFGレジスタ

0	0	CTL5=0	CTL4=0	CTL3=0	CTL2=1	CTL1=1	CTL0=1

アイドル時の状態設定
CTL[5:3]=‘0’
CTL[2:0]=‘1’

(**f**) GPIFIDLECTLレジスタ

〔図5〕**SRAMシングルリード用ウェーブフォームディスクリプタの設計**

LENGTH=0x01							
0	0	SGL=0	GINT=0	INCAD=0	NEXT=0	DATA=0	DP=0
LOGIC FUNCTION(未使用)							
0	0	CTL5=0	CTL4=0	CTL3=0	CTL2=1	CTL1=1	CTL0=0

(**a**) ステート#0

LENGTH=0x04							
0	0	SGL=0	GINT=0	INCAD=0	NEXT=0	DATA=0	DP=0
LOGIC FUNCTION(未使用)							
0	0	CTL5=0	CTL4=0	CTL3=0	CTL2=1	CTL1=0	CTL0=0

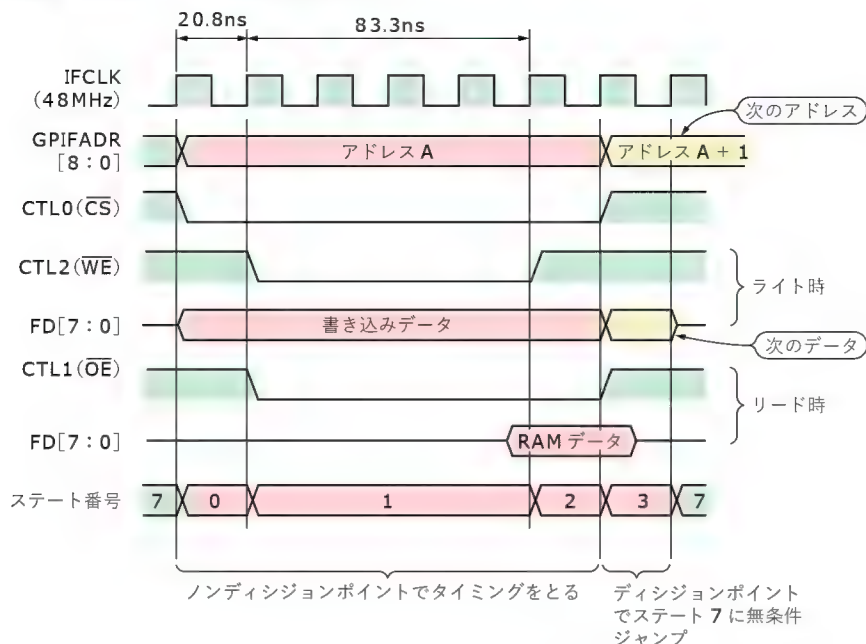
(**b**) ステート#1

LENGTH=0x01							
0	0	SGL=0	GINT=0	INCAD=0	NEXT=0	DATA=1	DP=0
LOGIC FUNCTION(未使用)							
0	0	CTL5=0	CTL4=0	CTL3=0	CTL2=1	CTL1=0	CTL0=0

(**c**) ステート#2

Re-Exec=0	0	BRANCHON1=0x07			BRANCHON0=0x07		
0	0	SGL=0	GINT=0	INCAD=1	NEXT=0	DATA=0	DP=1
LFUNC=00		TERMA=00			TERMB=00		
0	0	CTL5=0	CTL4=0	CTL3=0	CTL2=1	CTL1=1	CTL0=1

(**d**) ステート#3

〔図6〕SRAMバーストアクセスタイミング

を作り，それ以外のものは単純に割り込みの始末だけしてリターンさせました．

● サンプルファームウェアの割り込み処理

今回のサンプルは割り込み処理はEP0のみとし，バルクIN/OUTエンドポイントの処理に対しては，ポーリングで処理するようにしました．ベンダリクエストはEP0経由のアクセスになるので，割り込みで受けて処理します．

ベンダリクエストのうち，アドレスセットアップ要求，シングルライト要求，シングルリード要求は割り込みの中で完了させます．シングルリード/ライトに長い時間がかかる可能性がある場合は割り込み処理で行わず，フラグを立てるなどしてタスク側で処理させるようにすべきですが，今回は200nsもかからないので，割り込みの中で処理しました．

ベンダリクエストはいずれもアドレス情報をともないます．アドレスは全コマンド共通でwValuleフィールド(SETUPDAT[3:2]に入ってくる)にセットすることにしたので，とり出した値を，PA[7:4]，GPIFADRH，GPIFADRLに設定します．

シングルライト要求の実行はgpif_slgwt()関数をコールしていますが，これは単にGPIFがレディである(停止している)ことを確認してからXGPIFSGLDATLXにデータを書くことで，データのセットと同時にGPIFにキックをかけているだけです．今回はシングルライト動作は8クロック，160ns程度で終了してしまうので，終了をチェックせずにそのままリターンさせています．

シングルリード要求はgpif_sgld()関数によって処理しています．リード方向はまずGPIFにスタートをかけて，完了した後でデータを読むという動作になるので，シングルライトよりも一手間かかっています．GPIFがレディ状態にあるかどうかを確認した後でXGPIFXGLDATLXをダミーリードすることで，GPIFにキックをかけ，完了を待ってXGPIFSGLDATLNOXを読み出してラッチされたデータを引き取ります．もし，複数バイトの転送が必要ならば，ここで，XGPIFSGLDATLNOXのかわりにXGPIFSGLDATLXをリードすれば，前回の動作でラッチされたデータが読み出されるのと同時に次の転送動作が始まりますが，今回は1バイト転送しか行わないので，この機能は使っていません．

● サンプルファームウェアのタスク側の処理

ユーザータスク側ではバーストリード/ライト処理を行います．タスクのメインループの中では割り込み側でセットされるバーストリード要求サイズデータと，EP6(バーストリード用)のFULLフラグ，EP2(バーストライト用)のEMPTYフラグをチェックしています．

▶バーストリード要求処理

バーストリード要求サイズが0でないということは，ベンダリクエストでバーストリード要求が来たということなので，gpif_

〔図7〕バーストアクセス用ウェーブフォームディスクリプタの変更

Re-Exec＝0	0	BRANCHON1＝0x07			BRANCHON0＝0x07		
0	0	SGL＝0	GINT＝0	INCAD＝1	NEXT＝1	DATA＝0	DP＝1
LFUNC＝00		TERMA＝00			TERMB＝00		
0	0	CTL5＝0	CTL4＝0	CTL3＝0	CTL2＝1	CTL1＝1	CTL0＝1

(a) バーストライトステート#3(#0～#2はシングルライトと同じ)

Re-Exec＝0	0	BRANCHON1＝0x07			BRANCHON0＝0x07		
0	0	SGL＝0	GINT＝0	INCAD＝1	NEXT＝0	DATA＝0	DP＝1
LFUNC＝00		TERMA＝00			TERMB＝00		
0	0	CTL5＝0	CTL4＝0	CTL3＝0	CTL2＝1	CTL1＝1	CTL0＝1

(b) バーストリードステート#3(#0～#2はシングルリードと同じ)

burstrd()関数を呼び出してGPIFにキックをかけます．

ここではGPIFがレディ状態（停止状態）にあるかどうかを確認した後，EP2GPIFPFSTOP=0として，トランザクションカウンタ（GPIFの転送回数カウンタ）によって停止するように設定します．この後トランザクションカウンタ（EP6GPIFTCH/TCL）にホストからの要求サイズをセットしてEP6GPIFTRIGポートを読み捨てることで，GPIFにキックがかかりバーストリード動作が開始されます．トランザクションカウンタは転送回数カウンタなので，このサンプルのように8ビット単位の転送の場合には転送バイト数とトランザクションカウンタへの設定値は同じになりますが，16ビット単位の転送にする場合にはバイト数の半分の値を設定します．

GPIFがバーストリード動作を開始すると，自動的に指定サイズ分のデータがスレーブFIFOに格納され，転送が終了すると，スレーブFIFOはUSBエンドポイントバッファに自動的に切り替わります．今回はクァッドバッファモードで使っているので，この時点でまだ送らなければならないデータがあれば，さらに3パケット（合計4パケット）分まで先行して転送可能です．バッファにまだ空きがあるかどうか（4バンクのバッファを使いきっているかどうか）は，EP6CSレジスタのFULLフラグによって判定できるので，まだ送るデータがあり，FULLフラグが立っていなければ次の転送動作を行うということになります．

▶バーストライト要求処理

バーストライト側は，EP2CSレジスタのEMPTYフラグを見ています．1パケット分のデータが到達すると，EMPTYフラグが‘0’になるため，これをチェックして，バーストライト動作を行わせます（gpif_burstwt()関数）．こちらもバーストリードと同様にトランザクションカウンタをセットして，今度はEP2GPIFTRIGレジスタをライトすることで，GPIFにバーストライトのキックをかけます．この後，GPIFの転送動作終了を待ってメインルーチンに戻ります．

● コントロールプログラムの作成

ファームウェアのテストは，当初チップベンダが無償提供しているUSBコントロールパネルから行いました．ベンダIDとプロダクトIDをFX2のオリジナルのまま（ベンダID：0x04b4，プロダクトID：0x8613）にしておけば，コントロールパネルから，任意のベンダリクエストを発行したり，各エンドポイントのリード/ライトなどを自由に行えるので，各リクエストの処理チェックをする程度の簡単なテストはこれで十分です．

ただ，このままではいささか使い勝手が悪すぎるので，本誌でも何度か取りあげているUSB汎用ドライバとDLLを用い，Microsoftのサイトから無償で手に入るVisualBasic5.0-CCE（Control Creation Edition）で作成したアプリケーションからリード/ライトテストを行わせてみました．

サンプルはごく簡単なものですが，シングルリード/ライト，バーストリード/ライトの動作要求をかけることができます．

図8が今回作成したサンプルアプリケーションの画面です．

〔**図8**〕**FX2-SRAM接続テストアプリケーション画面**

Address欄にアクセスしたい先頭アドレスを16進数でセットします．この値はすべてのリード/ライト動作ともに共通です．アドレスを設定してシングルリードボタンを押せば，そのアドレスの値を1バイト読み出して横のデータ表示領域に表示します．

シングルライトボタンの下のRead Lengthテキストボックスは，バーストリードのデータ長を16進数で指定するものです．指定してから，下のBurstReadボタンを押すとAddressフィールドに書かれたアドレスからRead Lengthで指定されたサイズ分だけデータをリードして，16進ダンプ形式に整形して表示します．なお，今回はバッファサイズを8Kバイト取っているので，データサイズの最大値は0x2000になります．

Write Dataのテキストボックスは，シングルライトで書き込むデータを指定するものです．Single Writeボタンを押すと，Addressフィールドで指定されたアドレスにWrite Dataの値を1バイト書き込みます．

バーストライトはファイルからの転送を行わせるようにしました．左側で送りたいファイルを選択してBurst Writeボタンを押すと，指定されたファイルの内容をEP2（バルクOUTエンドポイント）に転送します．

● 転送性能の実測

作成したファームウェアと，VBアプリケーションによる転送を行った波形をロジックアナライザでとってみました．**図9**（**a**）および**図9**（**b**）がそれぞれシングルリード，シングルライト時の波形です．データバスがかなり後のほうで“H”に復帰しているのはハイインピーダンス状態になった後，プルアップ抵抗で“H”に復帰しているからで，とくに0xFFをドライブしているわけではありません．

設計では，リードパルス幅は5クロック（ステート1の4クロックとステート2の1クロック），ライトパルス幅は4クロック分としたので，計算上ではそれぞれ約104ns（1/48MHz × 5），83.3nsです．サンプリング5nsの実測で105ns〔**図9**（**a**）〕，85ns〔**図9**（**b**）〕なので，設計どおり動作していることがわかります．

リード時のデータはSRAMの公称アクセスタイムの100nsに比べてかなり早く出ています．

バーストライト時の波形が**図10(a)**です．1回のライトサイクルは，8サイクル(ステート0，1，2，3，7の合計)の設計なので，計算上約167nsに対してサンプリング5nsでの実測で165nsと，こちらも予定どおりの動きになっています．また，ステート2で$\overline{CS}$を立ちあげるのとともに，データバス上のデータやアドレスが進められているようすもよくわかります．

サンプリングを変更してバーストライト動作の全体像が見えるようにしたのが**図10(b)**です．帯状になっている部分が1パケット(512バイト)分の転送動作を行っているところで，帯と帯の間の隙間があいているように見える部分がGPIF動作が完了してから次の動作が開始されるまでのファームウェアによるオーバヘッドです．この間隔は12.60μsと読み取れます．今回のサンプルで512バイトの転送にかかる時間は計算上約85.3μsなので，1パケットあたり97.9μsとなり，約5Mバイト/秒となります．

同じようにバーストリードを測定したのが**図11**です．リード方向のときには転送サイズチェックとエンドポイントフラグのチェックなど少し手間がかかっているため，オーバヘッドが26.3μs〔**図11(b)**〕に増えています．このため，転送速度は約4.6Mバイト/秒に低下しています．

今回のサンプルではわかりやすさを優先したため，バースト完了をチェックしてリターンするようになっていますが，性能を稼ぐならGPIFにキックをかけたあと，次の転送準備をしてしまうことで，オーバヘッドを転送時間中に埋め込むほうがよいでしょう．

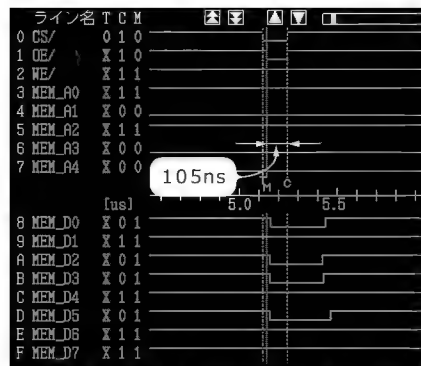
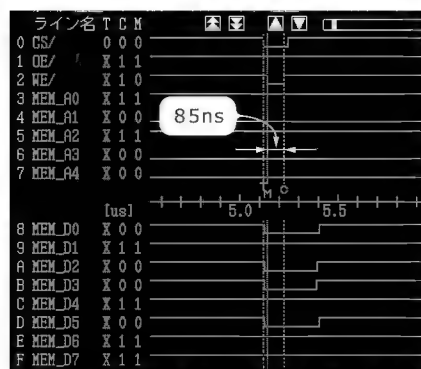
● 高速化の実験

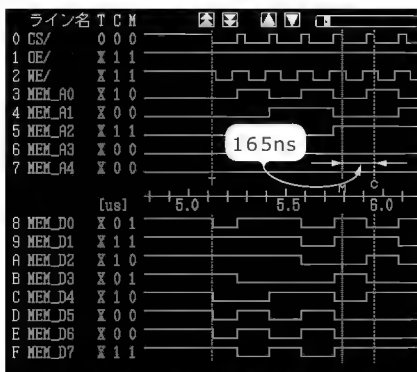
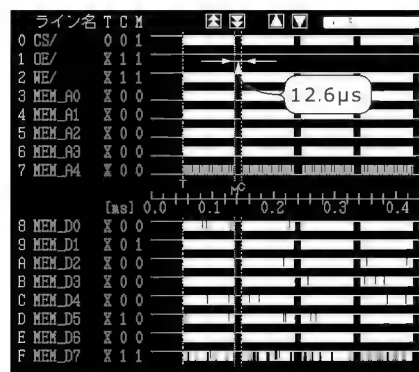
今回のサンプルではGPIFをバイトサイズで使っており，また1回の書き込みに8サイクル使っていることから，試しにステート1を4クロックから1クロックに減らし，さらにエンドポイントをワード(16ビット)として，8Kバイトのデータを転送してみたのが**図12**です．8Kバイトの転送に614.4μsかかっているので，平均伝送速度は約13Mバイト/秒(=104Mビット/秒)です．GPIFの1トランザクションあたりのステート数を減らしたり，ステートインストラクションのところで触れたRe-Executeモードを使うことなどにより，さらに性能をあげることも可能です．しかし一般的な使い方では，このくらいが転送速度の目安といってよさそうです．

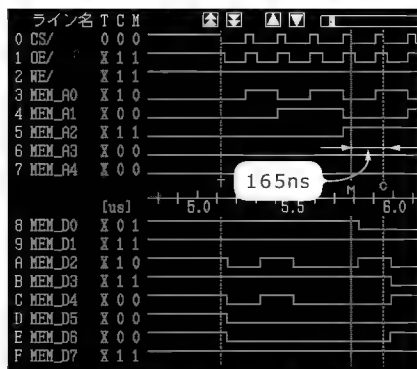
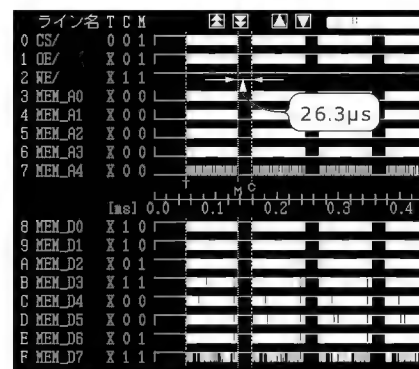
2 FX2同士のデータ転送の実験

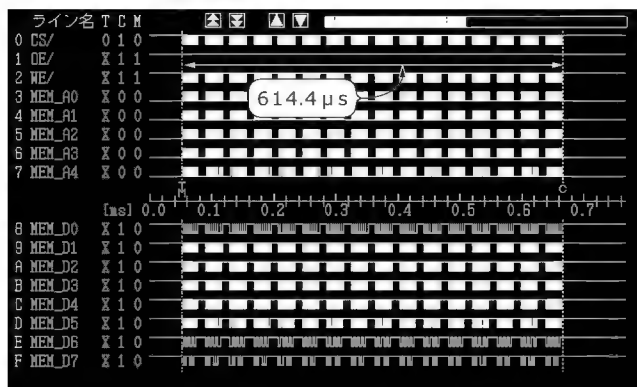
● FX2を対向で繋いでデータ転送

SRAM接続は入力を見て分岐するような部分がなかったので，もう少しステートマシンらしい動きを行わせる例として，FX2

〔図9〕シングルアクセス時の波形

(a) リードアクセス時(5nsサンプリング)

(b) ライトアクセス時(5nsサンプリング)

〔図10〕バーストライト時の波形

(a) 一部拡大(5nsサンプリング)

(b) 全体(50nsサンプリング)

〔図11〕バーストリード時の波形

(a) 一部拡大(5nsサンプリング)

(b) 全体(50nsサンプリング)

〔図12〕ステート1を1クロックに高速化し，8Kバイトのデータを転送している時の波形

同上を接続してハンドシェイクを使ったデータ伝送を行わせてみることにします．手順はSCSIなどでも使われているような，送信側からのストローブに対応して，相手がアクノリッジ信号を返すという方法を使ってみることにします．

接続関係は**図13**のように互いの16ビットGPIFデータバスを直結し，自身のCTL0を相手のRDY0とクロスにつなぐという，ごく単純なものです．

まず最初に，**図14**に示すようなシーケンスで行ってみました．

(1) 送信側がデータとともに$\overline{STB}$をアサート
(2) 受信側は$\overline{STB}$のアサートを検出したらデータバス上のデータを取り込み，$\overline{ACK}$をアサート
(3) 送信側は受信側の$\overline{ACK}$のアサートを検出したら$\overline{STB}$をネゲート
(4) 受信側は送信側の$\overline{STB}$のネゲートを検出したら$\overline{ACK}$をネゲート
(5) 送信側は受信側の$\overline{ACK}$ネゲートを検出したら(1)に戻る

これは，SCSIなどでも行われている，ごく一般的なハンドシェイク手順です．ただ，今回のようにFX2同士の対向で，しかも非同期にした場合にはRDY入力を2段ラッチモードにするよりないことから，伝送速度がかせげません．今回の例でも1回の伝送に15クロック(約310ns)ほどかかっています．

そこで，今回はこれを改造して**図15**のように，$\overline{STB}$の立ちあがりエッジも利用してデータ伝送を行うようにしてみました．SDRAMでいうところのDDR(Double Data Rate)と同じような考え方で，先程の手順で示すところの3)と4)でも，データの更新/取り込みを行うことで，ほぼ2倍の伝送速度をかせごうというものです．

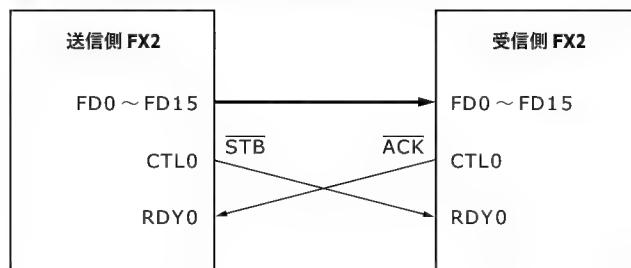
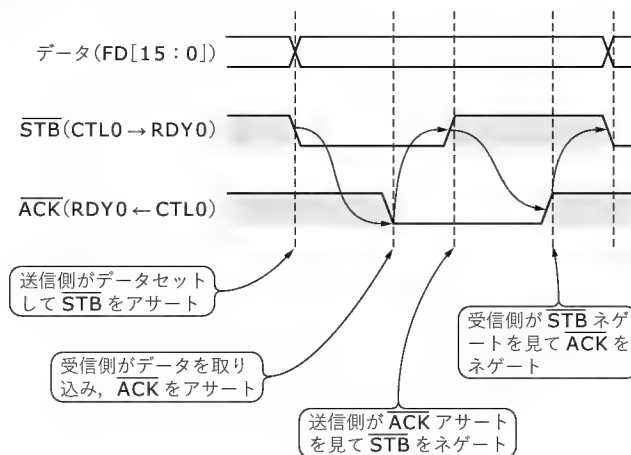
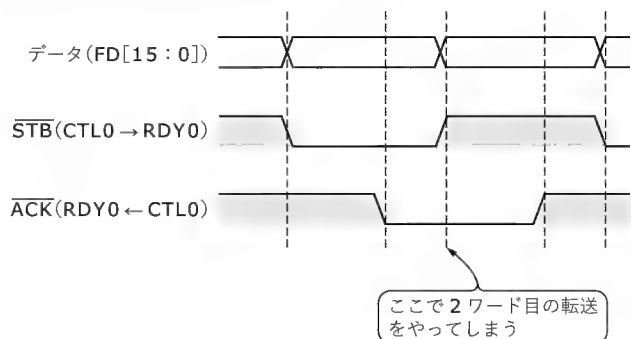
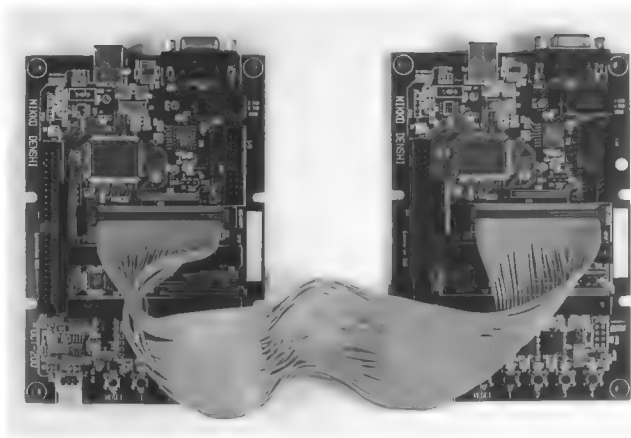
試作ボードのようすを**写真2**に示します．

● FX2間伝送プログラムの作成

FX2間の手順が決まったので，次にソフトウェアの変更を考えます．

▶伝送実験の構成

伝送実験にはUSB2.0ホストブリッジを搭載したホストPCが2台あればよいのですが，手持ちの都合で1台しか用意できなか

〔図13〕FX間データ転送テスト用接続

〔図14〕基本的なハンドシェイク手順

〔図15〕FX2間バースト伝送ハンドシェイク

〔写真2〕FX2対向通信テストのようす

ったため，今回のテストでは**図16**のように，1台のPCに刺された USB2.0ホストアダプタの二つのポートにそれぞれFX2ボードを接続して行いました．

同じIDのボードが二つあると混乱するので，FX2のデフォルトのIDと先ほどのSRAM接続アプリケーション用のIDを利用します．

- 送信側はCypressのコントロールパネルを使ってEP2にバルクOUTする
- 受信側はSRAM接続テストで作ったVisualBasicのアプリケーションを利用してベンダリクエストを行いEP6をリード，データ表示を行う

ファームウェアは送信側，受信側を分けて作るのも面倒なので，SRAMアクセス用のプログラムを流用し，内部の処理はまったく同じものにしておき，ベンダID，プロダクトIDをコンパイル時のオプション指定で切り替えることで，ベンダID，プロダクトIDが違うだけのファームウェアを作成するようにします．

▶ウェーブフォームの設計

ウェーブフォームディスクリプタをいきなり書き下すのはなかなか面倒なので，まずタイミング図を書き，それにあわせてステートを割り付けていくことにしましょう．今回は相手からの入力もあり得るので，内部の2段ラッチのことにも配慮しながら書いたのが**図17**です．実際には送信側と受信側は別クロックで動いているので，クロックの位相関係によっては平均0.5クロックのずれは生じますが，ここでは簡単のために送受信側とも同一周波数，同一位相のクロックで動いているものとします．さて，これを簡単に順を追いながら説明しておきましょう．

(1) 送信側はステート0でデータを制定させるのとともに$\overline{STB}$(CTL0出力)を“L”に落とす(クロック1)

(2) 受信側で$\overline{STB}$が二段ラッチされてGPIFに取り込まれ，$\overline{STB}$=“L”が検出されるとステートが1に移動する．データ取り込みとともに$\overline{ACK}$を“L”にする(クロック4)

(3) 送信側で$\overline{ACK}$が二段ラッチされてGPIFに取り込まれ，$\overline{ACK}$=“L”が検出されると，ステート1に移動する．データ更新とともに$\overline{STB}$を“H”に戻す(クロック7)

(4) 受信側で$\overline{STB}$=“H”が検出され，データラッチとともに$\overline{ACK}$を“H”に戻す．(クロック10)．この後，受信側はステート7に無条件ジャンプ(クロック11)

(5) 送信側は$\overline{ACK}$=“H”が検出されるとステート2に移動し，データ更新(クロック14)．この後，送信側ステートはステート7に無条件ジャンプ(クロック15)

図からわかるとおり，1回の転送で15クロックかかることになります．1回の伝送で送っているデータは4バイトなので，GPIFを48MHzで動作させれば，バス上のデータ転送速度は，48 ÷ 15 × 4=12.8Mバイト/秒となります．ソフトウェアのオーバヘッドを入れて10Mバイト/秒弱程度というところでしょう．

これに基づいてSRAMの例と同じようにバーストリード/ライトのウェーブフォームディスクリプタを設計していけばよいわけです．

▶プログラムの変更

プログラムのほうは，バーストリード/ライトの波形が変わっ

〔図16〕FX2間の伝送テスト構成

〔図17〕FX2間の伝送手順とステート割り付け

ても，スレーブFIFOやGPIFの操作方法が変わるわけではないので，処理自体に大きな違いはありません．おもな違いは，

- 初期化のときにGPIFのバスを16ビット幅で使うためのレジスタの値を変更する
- RDY入力を非同期入力モード（2段ラッチモード）に設定する
- GPIFのレディ待ちのタイムアウト処理をはずす

の3点です．

まず，データ伝送をワード（16ビット）幅で行うため，EP2FIFOCFG，EP6FIFOCFGの最下位ビット（WORDWIDE）を‘1’にします．さらにGPIFのRDY入力を2段ラッチにするために，GPIFREADYCFG.6（SASビット）を‘0’にしておきます．先に触れたように，マニュアルではこのビットは‘1’で非同期入力（2段ラッチ）となっていますが，実際の波形を見る限り，これは間違いのようです．

また，伝送プログラムは相手からの出力信号がない限り，GPIFは停止しています．このため，`wait_gpif_ready()`関数の中のGPIFのレディ待ちのタイムアウト処理を削除して，レディになるまで無限ループするようにしておきました．

● 伝送実験

変更が終わったので，コンパイルしてダウンロードします．ダウンロードするとき，2枚のFX2ボードを同時に挿しておいても悪くはないのですが，まず受信側を挿入してファームウェアをダウンロードして，FX2のオリジナルと別IDで再認識させた後で，送信側のボードを挿して送信側ファームウェアをダウンロードするという手順がわかりやすいかもしれません．

ダウンロードが終わったら，まずVisualBasicアプリケーション側でサイズを指定してバーストリードボタンを押して受信待ち状態にしておきます．

EZ-USBコントロールパネルで送信側のFX2ボードのEP2（OUTエンドポイント）に受信側の指定サイズ分の大きさのファイルを選択して，それを送ります．

● 転送性能の実測

4Kバイトのインクリメントデータを使って波形を取ったのが**図18（a）**です．インクリメントデータをワード転送しているので，D0はいつも“L”になってしまっています．

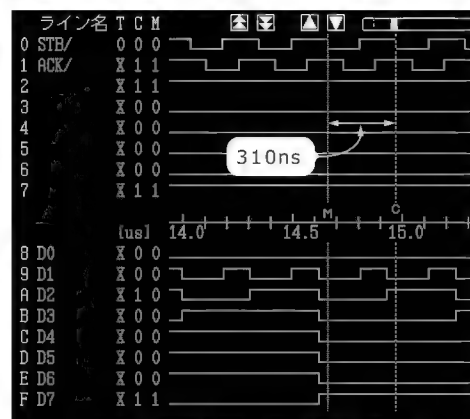
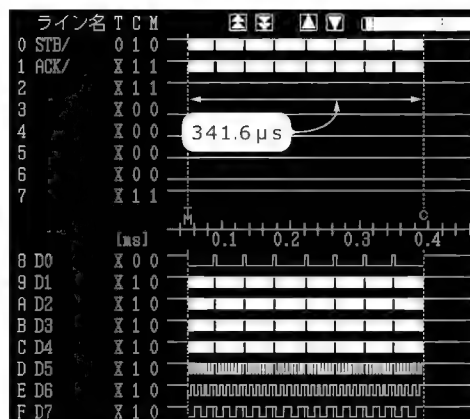
図を見ると1サイクルが約310nsなので，GPIFの動作クロック（48MHz）にして15クロックサイクルかかっています．設計どおりというところです．また，データバスの動きを見ると，確かに1サイクルで2バイト伝送していることがわかります．

また，サンプリングを荒くして全体を見たのが**図18（b）**です．細切れになっている一つずつが1パケット分（512バイト）の伝送です．全体で341.6μsなので，約11.7Mバイト/秒の伝送速度が得られています．

〔図18〕4Kバイトのインクリメントデータ転送波形

（a）一部拡大（5nsサンプリング）

（b）全体（50nsサンプリング）

おわりに

GPIFとスレーブFIFO機能を使ったごく基本的なデータ伝送を行ってみましたが，いかがだったでしょうか？　GPIF，スレーブFIFOとももっている機能は非常に多く，限られたページ数ではとても説明しきれないので，興味をもたれた方はぜひFX2のマニュアルを読まれることをお勧めします．

USB2.0対応となったEZ-USB FX2はEZ-USBファミリの特徴を受け継いだ，扱いやすいコントローラです．とくにFX2の大きな特徴であるGPIFは，FX2を使った各種伝送アダプタを簡単に作成できる可能性を感じさせるものであるといえるでしょう．

480Mbpsという数値がひとり歩きしてしまい，公称値のわりに性能が出ないなどといわれるUSB2.0ですが，これだけ単純なハードウェアと簡単なソフトウェアで100Mビット/秒もの速度で伝送が行えるということは，評価されてもよいと思います．

なお今回のサンプルのベンダIDには，来栖川電工の0x0b7eを使用させていただきました．この場を借りてお礼申し上げます．

参考文献

1）TECH I Vol.8，『USBハード＆ソフト開発のすべて』，CQ出版（株）
2）桑野雅彦，「USB2.0対応USB学習キット新登場！」，Interface，2001年11月号

くわの・まさひこ　パステルマジック

〔リスト1〕usrtask.c

```
//==================================================
//==                                              ==
//==     FX2(AN68013) デバイスファームウェア            ==
//==                                              ==
//==         ユーザータスク                          ==
//==                                              ==
//==         USB2<=>SRAM リード/ライトテスト           ==
//==             for "Interface" CQ Pub.           ==
//==                                              ==
//==     マスストレージクラス処理を外した                 ==
//==     単純なリード/ライトテストサンプル                 ==
//==                                              ==
//==     2002-12-24:UCT-200向けUSB/SCSI変換ベースに改造  ==
//==     2002-12-28:FX2向けに汎用性を持たせる            ==
//==             ・VID&PID変更の切り口                 ==
//==             ・レジスタ初期化                      ==
//==             ・割り込みフック                      ==
//==              など                              ==
//==     VID:PID                                  ==
//==     04B4:8613 (コントロールパネル用)              ==
//==     0B7E:800F (VBサンプル用仮ID)                 ==
//==                                              ==
//== Written by M.Kuwano (PastelMagic)            ==
//==                                              ==
//==================================================

#include "struct.h"
#include "stdef.h"
#include "fx2regs.h"
#include "f_fx2sram.h"
#include "f_gpif.h"

//WORD   vid = 0x04b4;
//WORD   pid = 0x8613;
WORD     vid = 0x0b7e;
WORD     pid = 0x800f;

BYTE     gstatus;  // Global Status :今回は使ってない

WORD     inxfrlen; // EP2-INに転送して欲しいデータサイズ

//
// レジスタ初期化
//
void usr_reginit(void)
{
    // GPIFは内部48MHzで動作 GPIFマスタモードで使う
    IFCONFIG  = 0xc2;
    // 念のためGPIFも止めておく
    GPIFABORT = 0;

    // セットアップデータ割り込み，ハイスピード，バスリセット割り込みイネーブル
    USBIE     = USBIE_SUDAV | USBIE_HSGRANT | USBIE_USBRES;
    // INT4はFIFO/GPIF，オートベクタ機能は使わない
    INTSETUP = INTSETUP_INT4SRC;
    // EP2とEP6割り込みを使う
    // EPIE       = EPIE_EP6 | EPIE_EP2;
    // 今回のサンプルはポーリング処理にするのでエンドポイント割込みは使わない
    EPIE  = 0;

    // バルクOUT，512×4バンク
    EP2CFG    = EP2CFG_VALID | EP2CFG_TYPE_BULK | EP2CFG_BUF_QUAD;
    // バルクIN，512×4バンク
    EP6CFG    = EP6CFG_VALID | EP6CFG_DIR | EP6CFG_TYPE_BULK |
                                             EP6CFG_BUF_QUAD;
    // とりあえず書いておく
    EP0CS     = EP0CS_HSNAK;
    FIFORESET = 0x80;
    FIFORESET = 0x82;
    FIFORESET = 0x86;
    FIFORESET = 0x00;
    // Quad Bufferなのでとりあえず4回書いて返しておく
    EP2BCL          = 0x80;
    EP2BCL          = 0x80;
    EP2BCL          = 0x80;
    EP2BCL          = 0x80;
    EP2CS                  = 0;

    // Zero-Length Packet イネーブル，FIFOはバイトサイズ
    EP2FIFOCFG      = 0x04;
    //
    EP6FIFOCFG      = 0x04;

    // とりあえずDATA0から始まるようにしておく
    TOGCTL          = 0x16;
    TOGCTL          = 0x36;
    // とりあえずDATA0から始まるようにしておく
    TOGCTL          = 0x02;
    TOGCTL          = 0x22;

    // GPIF完了割り込み
    GPIFIE          = GPIFIE_GPIFDONE;
    // PAはI/Oポートとして使用する (PA[7:4]=MEM_ADRS[12:9]
    PORTACFG        = 0x00;
    // とりあえずPA[7:4]のみドライブ
    OEA         = 0xf0;
    // PA[7:4]='0000'にしておく
    IOA         = 0x00;
    // READY入力は非同期，2段ラッチ (今回はREADY入力は使わないので関係ない)
    GPIFREADYCFG    = GPIFREADYCFG_SAS;
    // CTL[5:3]='L',CTL[2:0]='H'
    GPIFIDLECTL     = 0x07;
    // すべてトーテムポール出力にする
    GPIFCTLCFG      = 0x00;
    // アイドル時にはデータバスをドライブしない
    GPIFIDLECS      = 0x00;
    // デフォルトのまま
    GPIFWFSELECT    = 0xe4;
    // おまじない
    GPIFREADYSTAT    = 0x11;

    // PORTCはGPIFADR[7:0]として使う
    PORTCCFG  = 0xff;
    // GPIFADRとして使うので無効だが一応設定だけ
    OEC         = 0x00;
    // PORTE[7]はGPIFADR[8]として使用
    PORTECFG  = 0x80;
    // とりあえずドライブしない
    OEE         = 0x00;

    // 一応クリアしておく
    GPIFADRH  = 0x00;
    GPIFADRL  = 0x00;

    // AUTOモードは使わないけど，一応設定だけ
    EP2AUTOINLENH = ep26size >> 8;
    EP6AUTOINLENH = ep26size >> 8;
    EP2AUTOINLENL = ep26size & 0xff;
    EP6AUTOINLENL = ep26size & 0xff;

    // Programmable Flagは使わないが一応
    EP2FIFOPFH     = 0x80;
    EP4FIFOPFH     = 0x80;
    EP6FIFOPFH     = 0x80;
    EP8FIFOPFH     = 0x80;
    EP2FIFOPFL     = 0x00;
    EP4FIFOPFL     = 0x00;
    EP6FIFOPFL     = 0x00;
    EP8FIFOPFL     = 0x00;
    EP2GPIFPFSTOP = 0x00;
    EP4GPIFPFSTOP = 0x00;
    EP6GPIFPFSTOP = 0x00;
    EP8GPIFPFSTOP = 0x00;
}

//
// ユーザー初期化ルーチン
// これが終わった後に割り込みがイネーブルになる
//
void usr_init(void)
{
    waveset_singlewt();
    waveset_singlerd();
    waveset_burstwt();
    waveset_burstrd();
    gstatus = READY;
    inxfrlen = 0;
}

//
// ユーザータスク
//
void usr_task(void)
                                    ～以下省略～
```

〔リスト2〕gpif.c

```
//====================================================
//==          GPIF 設定&コントロール                  ==
//==                                                ==
//==     アドレスバスの設定:    gpif adrsset(WORD adrs) ==
//==     シングルライト:   gpif singlewt(BYTE data)    ==
//==     シングルリード:   gpif singlewt()             ==
//==     バースト(FIFO)ライト: gpif burstwt()          ==
//==     バースト(FIFO)リード: gpif burstrd()          ==
//==                                                ==
//==     gpif adrsset は PORTA[7:4]/GPIFADRH/GPIFADRL  ==
//==     にアドレスを設定                              ==
//==     gpif singlewt/singlerd は1バイトのライト/リード ==
//==     を行う.                                      ==
//==                                                ==
//==                                                ==
//==                                                ==
//==                                                ==
//====================================================

// PA[7] : SRAM A[12]
// PA[6] : SRAM A[11]
// PA[5] : SRAM A[10]
// PA[4] : SRAM A[09]
// PA[3] : N.U.
// PA[2] : N.U.
// PA[1] : N.U.
// PA[0] : N.U.
//
// PB[7]/FD7  : SRAM D7
// PB[6]/FD6  : SRAM D6
// PB[5]/FD5  : SRAM D5
// PB[4]/FD4  : SRAM D4
// PB[3]/FD3  : SRAM D3
// PB[2]/FD2  : SRAM D2
// PB[1]/FD1  : SRAM D1
// PB[0]/FD0  : SRAM D0
//
// PD[7]/FD15 : N.U
// PD[6]/FD14 : N.U
// PD[5]/FD13 : N.U
// PD[4]/FD12 : N.U
// PD[3]/FD11 : N.U
// PD[2]/FD10 : N.U
// PD[1]/FD9  : N.U
// PD[0]/FD8  : N.U
//
// CTL5       : N.U.
// CTL4       : N.U.
// CTL3       : N.U.
// CTL2       : -SRAM WE
// CTL1       : -SRAM OE
// CTL0       : -SRAM CS
//

// RDY5       : N.U.
// RDY4/PA0   : N.U.
// RDY3       : N.U.
// RDY2       : N.U.
// RDY1       : N.U.
// RDY0       : N.U.

#include "stdef.h"
#include "struct.h"
#include "fx2regs.h"
#include "f usrtask.h"

//
// GPIFWFSELECT = 0xe4 (11 10 01 00)
//
#define  WF BURSTRD     0
#define  WF BURSTWT     1
#define  WF SINGLERD    2
#define  WF SINGLEWT    3

//
// GPIF が READY 状態 (動作完了状態) になるまで待つ
//
BYTE wait gpif ready()
{
    WORD i;
    for (i=0; i<0xf000; i++) {
         if (GPIFIDLECS & GPIFIDLECS DONE)
              break;
    }
    if (i>= 0xf000) {
         gstatus = E GPIF NOT READY;
         return(ERROR);
    }
    return(READY);
}

//
// GPIF 強制停止
//
void abort gpif()
{
    if (GPIFIDLECS & GPIFIDLECS DONE)      // すでに止まっている
         return;
    EP2FIFOPFH = 0x80;
    EP6FIFOPFH = 0x80;
    EP2FIFOPFL = 0;
    EP6FIFOPFL = 0;
    EP2GPIFFLGSEL = 0;
    EP6GPIFFLGSEL = 0;
    EP2GPIFPFSTOP = 1;
    EP6GPIFPFSTOP = 1;
    EP2GPIFPFSTOP = 0;
    EP6GPIFPFSTOP = 0;
}

BYTE chk gpif()
{
    if (wait gpif ready() != READY) {
         abort gpif();
         if (wait gpif ready() != READY) {
              return(ERROR);
         }
    }
    return(READY);
}

void wavemem set(xdata BYTE *src, xdata BYTE *dst)
{
    xdata BYTE *sp,*dp;
    BYTE c,verify ok;
    chk gpif();
    sp = src + 0x1f;
    dp = dst + 0x1f;
    for (c=0; c<0x20; c++) {      // ゼロクリア
         *dp-- = 0;
    }
    sp = src + 0x1f;
    dp = dst + 0x1f;
    for (c=0; c<0x20; c++) {
         if (*sp != 0)
              *dp = *sp;
         dp--;
         sp--;
    }
    do {
         dp = dst + 0x1f;
         sp = src + 0x1f;
         verify ok = TRUE;
         for (c=0; c < 0x20; c++) {
              if (*sp != *dp) {
                   *dp = *sp;
                   verify ok = FALSE;
                   break;
              }
              sp--;
              dp--;
         }
    } while(verify ok == FALSE);
}

//
// アドレスセットアップ
//
//
// 今回のサンプルでは
// A[15:13]:未使用
// A[12:9]  : PA[7:4]
// A[8]     : GPIFADRH
```

～以下省略～

Chapter 3

組み込み機器にUSB周辺機器を接続するために

USBホストコントローラの概要とプロトコルスタックの移植

芹井滋喜

組み込み機器にUSBを採用する場合，これまではUSBターゲット機器として設計されることが多かった．しかし今後は，自分がホストになってUSB周辺機器を接続して制御したいという要求も増えてくるだろう．ここではOHCIに準拠したUSBホストコントローラを内蔵したSH7727(SH3-DSP)を使い，市販されているUSBホストのプロトコルスタックを移植した事例を解説する．

（編集部）

はじめに

● USBドライバと一口にいっても……

最近ではUSBがかなり普及してきた関係で，USBドライバを実際に作られた方や，USBドライバのコーディングに関する記事などの資料を読まれたことのある方はかなり多くなっていると思います．しかし，このような形でUSBドライバを経験された方は，USBデバイス用のクライアントドライバである例が大半で，ホストドライバを経験されている人はほとんどおられないのではないでしょうか．

じつのところ，筆者もいままではクライアントドライバしか扱ったことがありませんでした．ドライバの開発は，やはりWindows用のドライバが圧倒的に多いのですが，Windowsの場合，すでにホストドライバが用意されているので，実際にホストドライバを作ることは，USBホストコントローラの開発でもしないかぎり必要のない作業です．

ところが最近では，USBデバイスはWindows以外でも使われることが多くなってきました．PDAなどでもUSBが使えるものが増えてきているし，ディジタルカメラとプリンタをUSBで直接接続できれば，パソコンがなくても印刷できて便利です．

● 組み込み機器におけるUSBホストドライバ

このような機器で使用されるOSはWindowsではなく，組み込み用のOSとなります．組み込み用のOSの場合には，必ずしもUSBのホストドライバが用意されているわけではありません．このような場合は，新規にホストドライバを用意するか，ミドルウェアとして販売されているホストドライバを購入して使用することになります．USBのホストドライバをまったく新規に作るのは，あまり現実的ではないので，ほとんどの場合，ミドルウェアを購入して，ターゲットのハードウェア用に移植を行うことになります．

● SH7727とFlexiStack

今回，幸運にも，「FlexiStack」というミドルウェアの移植を行う機会に恵まれました．本章では，その移植事例について詳しく解説します．

FlexiStackは，Philips社が自社のUSBホストコントローラLSI用に開発・販売しているミドルウェアで，日本では(株)スティルが販売しています．今回はこのFlexiStackを，SHシリーズのCPUでUSBホストコントローラを搭載しているSH7727に移植を行いました．OSにはμITRONを使用しました．

それではまず，SH7727について解説したあと，FlexiStackの概要と移植作業について解説します．

1 SH7727とSolutionEngine

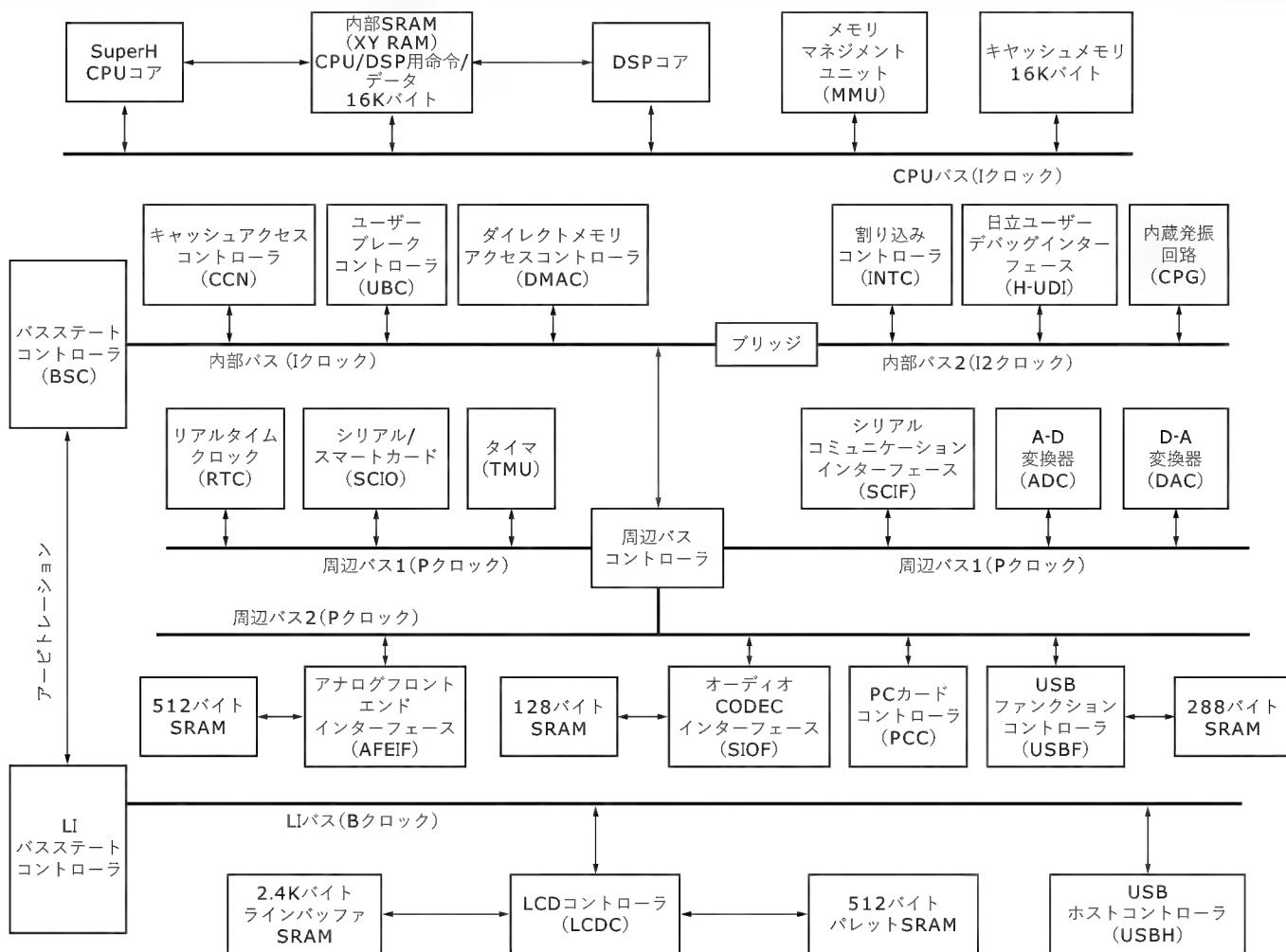
今回移植を行ったハードウェアは，SH7727を搭載した評価ボード“SolutionEngine”(MS7727SE01)です．SH7727はSH3-DSPをコアにもつCPUで，最大160MHzのクロックで動作します．また，このCPUはさまざまな周辺回路を内蔵しています．

今回このCPUを採用したのは，このCPUがUSBのホストコントローラを内蔵しているためです．またSolutionEngineは，日立製作所がSH7727の評価用に販売しているもので，SH7727を搭載し，SH7727のもつ機能をひととおりテストできる必要十分な周辺装置を備えています．今回のような，ソフトウェアの評価や製品開発前のソフトウェア開発には，非常に便利なボードです．

● SH7727のハードウェア

図1にSH7727のブロック図を示します．DSP機能をもつSH3-DSPをコアに，周辺機能を取り込んだCPUとなっています．SH7727のおもな特徴は次のとおりです．

- 動作周波数：100/160MHz
- CPU性能　：130/208MIPS
- キャッシュ：16Kバイト
- USB1.1対応のファンクション/ホスト内蔵
- LCDコントローラ内蔵により液晶表示専用のフレームバッフ

〔図1〕SH7727のブロック図

ァが不要

- 6万5千色のVGA液晶表示が可能
- 携帯端末に必要な機能をすべて1チップで実現

SH7727の応用例としては，HPC，Palm PC，WebPhone，POS端末，インターネット端末などがあります．実際，USBホストコントローラだけでなく，DSPコア，A-D/D-Aコンバータ，スマートカードインターフェースなどの周辺装置が内蔵されているので，いろいろ面白い応用製品が考えられそうです．

● USBホストコントローラ――OHCIとUHCI

現在，USB1.1のホストコントローラには，OHCI(Open Host Controller Interface)方式とUHCI(Universal Host Controller Interface)方式の2種類があり，それぞれ制御方法が異なります．

UHCIはIntelが主導して策定したUSB規格であり，OHCIはMicrosoft，National Semiconductorなどが中心となって策定した規格です．Windowsではそれぞれの方式のドライバが付属しているので，ユーザーがこの二つの方式を意識することはほとんどありません．

実際には，IntelとVIAがUHCI，それ以外のほとんどのメーカーはOHCIを採用しているようです．今回使用したSH7727も，OHCI準拠のホストコントローラを搭載しています．

UHCI方式は，インターフェースがシンプルなためコストはかかりませんが，その分だけドライバソフトに負荷がかかります．それに対してOHCI方式は，バスマスタ転送をサポートしているため，CPUにあまり負荷をかけないという特徴があります．

Windowsのように，標準で両方の方式のUSBホストドライバが用意されている場合は，ユーザーがこの二つの違いを意識することはほとんどありません．しかし，今回のように組み込み機器にUSBホストを実装する場合には，どちらのコントローラを選定するかが重要になります．SH7727はOHCI準拠のホストコントローラであり，FlexiStackもOHCIをサポートしているので，移植はかなり簡単に行えます．

● SH7727のUSBホストコントローラのレジスタセット

すでに説明したように，SH7727にはOHCI準拠のUSBホス

〔表1〕USBホストコントローラのレジスタセット一覧

名　称	R/W	初期値	アドレス
HcRevision	R	00000010h	04000400h
HcControl	R/W	00000000h	04000404h
HcCommandStatus	R/W	00000000h	04000408h
HcInterruptStatus	R/W	00000000h	0400040ch
HcInterruptEnable	R/W	00000000h	04000410h
HcInterruptDisable	R/W	00000000h	04000414h
HcHCCA	R/W	00000000h	04000418h
HcPeriodCurrentED	R/W	00000000h	0400041ch
HcControlHeadED	R/W	00000000h	04000420h
HcControlCurrentED	R/W	00000000h	04000424h
HcBulkHeadED	R/W	00000000h	04000428h
HcBulkCurrentED	R/W	00000000h	0400042ch
HcDonreHeadED	R/W	00000000h	04000430h
HcFmInterval	R/W	00002EDFh	04000434h
HcFrameRemaining	R	00000000h	04000438h
HcFmNumber	R	00000000h	0400043ch
HcPeriodicStart	R/W	00000000h	04000440h
HcLSThreshold	R/W	00000628h	04000444h
HcRhDescriptorA	R/W	TBD	04000448h
HcRhDescriptorB	R/W	TBD	0400044ch
HcRhStatus	R/W	00000000h	04000450h
HcRhPortStatus1	R/W	TBD	04000454h
HcRhPortStatus2	R/W	00000000h	04000458h

トコントローラを内蔵しています．**表1**に，USBホストコントローラのレジスタとアドレスの一覧を示します．それぞれのレジスタの機能は**表2**のようになっています．この中でも重要なのは，

- HcRevision
- HcControl
- HcCommandStatus
- HcInterruptStatus
- HcPeriodCurrentED
- HcControlHeadED
- HcControlCurrentED
- HcFmInterval
- HcFrameRemaining
- HcFmNumber

の各レジスタでしょうか．これらのレジスタのビット割り当てのようすを，**図2**(pp.86-87)に示します．

● SolutionEngineのハードウェア

次にSH7727の評価ボードである，SolutionEngine(MS7727SE01)のハードウェアについて説明します．**写真1**(p.87)にSolutionEngineの外観を示します．また，**図3**に各部の説明図を示します．

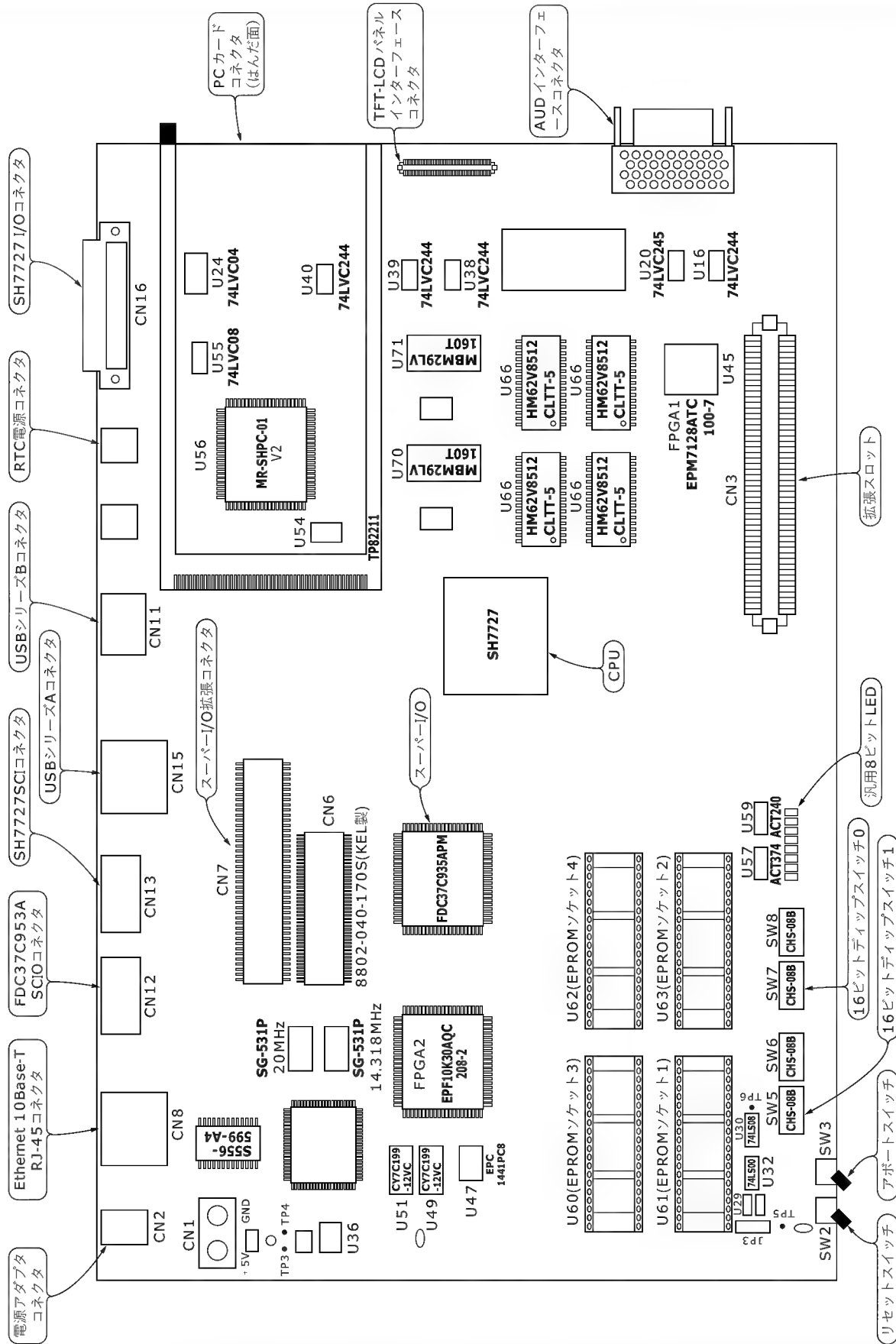
図4(p.88)にSolutionEngineのブロック図を示します．このボードには，SH7727の動作や性能の評価をするのに，必要十分な各種周辺装置が搭載されています．ブロック図でわかるように，USBホストのほかに，LCDインターフェースやRS-232-C,

〔表2〕USBホストコントローラの各レジスタ概要

レジスタ名	機　能
HcRevisionレジスタ	HCIスペシフィケーションのバージョン(BCD表現)．現在の値は10hでバージョン1.0を示す
HcControlレジスタ	ホストコントローラの操作モードを定義
HcCommandStatusレジスタ	ホストコントローラの現在のステータスを反映するだけでなく，ホストコントローラドライバにより発行されたコマンドを受け取るために使用される
HcInterruptStatusレジスタ	ハードウェア割り込みを起こすさまざまなイベントにおいてステータスを示す
HcInterruptEnableレジスタ	このレジスタの各イネーブルビットは，HcInterruptStatusレジスタの関連した割り込みビットに相当
HcInterruptDisableレジスタ	このレジスタの各ディセーブルビットは，HcInterruptStatusレジスタの関連した割り込みビットに相当
HcHCCAレジスタ	ホストコントローラコミュニケーションエリアの物理アドレスを示す
HcPeriodCurrentEDレジスタ	現在のIsochronous EDあるいはInterrupt EDの物理アドレスを示す
HcControlHeadEDレジスタ	コントロールリストにおいて最初のEDの物理アドレスを示す
HcControlCurrentEDレジスタ	コントロールリストにおいて現在のEDの物理アドレスを示す
HcBulkHeadEDレジスタ	バルクリストの最初のEDの物理アドレスを示す
HcBulkCurrentEDレジスタ	バルクリストにおいて現在のEDの物理アドレスを示す
HcDoneHeadEDレジスタ	Done queueに付加された，完了されたTDの物理アドレスを示す
HcFmIntervalレジスタ	フレームのビットタイム間隔(すなわち二つの連続的なSOF間)を示す14ビット値と，scheduling overrunを起こさずにホストコントローラが送受信するフルスピードでの最大パケットサイズを15ビットで示す
HcFrameRemainingレジスタ	現在のフレームに残っているビットタイムを示す14ビットのダウンカウンタ
HcFmNumberレジスタ	ホストコントローラとホストコントローラドライバにおいて起こるイベント間のタイミングの参照を示す16ビットカウンタ
HcPeriodicStartレジスタ	ホストコントローラが周期的なリストを処理しはじめるべきであるもっとも早い時間を決定するレジスタ
HcLSThresholdレジスタ	EOFの前に最大8バイトのLSパケットの転送に委任するかどうかを決めるため，ホストコントローラにより用いられるレジスタ
HcRhDescriptorA/Bレジスタ	ルートハブの仕様を示すレジスタ
HcRhStatusレジスタ	下位16ビットはハブステータスビットを表し，上位16ビットハブステータスチェンジビットを示す
HcRhPortStatus1/2レジスタ	ポートごとのベース制御とポートイベントの報告に使われる．上位ワードがステータス変化を反映し，下位ワードはポートステータスを反映する

〔図3〕SolutionEngineの各部概要

Information ── 東芝とアルパイン，車載情報機器事業会社を設立

(株)東芝とアルパイン(株)は，車載通信ゲートウェイシステムをはじめとする車載情報機器の商品企画，研究開発，製造，販売，アフターサービスを行う合弁会社である東芝アルパイン・オートモティブテクノロジー(株)を設立した．

〔図2〕USBホストコントローラの各レジスタ構造（一部のレジスタのみ）

▶ HcRevisionレジスタ

ビット：	31															16
	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	16
初期値：	0															0
R/W：	R															R

ビット：	15							8	7							0
	–	–	–	–	–	–	–	–	Rev							
初期値：	0							0	0	0	0	1	0	0	0	0
R/W：	R							R	R							R

▶ HcControlレジスタ

ビット：	31															16
	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	16
初期値：	0															0
R/W：	R/W															R/W

ビット：	15				11	10	9	8	7	6	5	4	3	2	1	0
	–	–	–	–	–	RWE	RWC	IR	HCFS1	HCFS0	BLE	CLE	IE	PLE	CBSR1	CBSR0
初期値：	0				0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R/W				R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

▶ HcCommandStatusレジスタ

ビット：	31													18	17	16
	–	–	–	–	–	–	–	–	–	–	–	–	–	–	SOC1	SOC0
初期値：	0													0	0	0
R/W：	R/W													R/W	R/W	R/W

ビット：	15											4	3	2	1	0
	–	–	–	–	–	–	–	–	–	–	–	–	OCR	BLF	CLF	HCR
初期値：	0											0	0	0	0	0
R/W：	R/W											R/W	R/W	R/W	R/W	R/W

▶ HcInterruptStatusレジスタ

ビット：	31	30	29													16
	–	OC	–	–	–	–	–	–	–	–	–	–	–	–	–	–
初期値：	0	0	0													0
R/W：	R/W	R/W	R/W													R/W

ビット：	15								7	6	5	4	3	2	1	0
	–	–	–	–	–	–	–	–	–	RHSC	FNO	UE	RD	SF	WDH	SO
初期値：	0								0	0	0	0	0	0	0	0
R/W：	R/W								R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

▶ HcHCCAレジスタ

ビット：	31		16
		HCCA	
初期値：	0	0	
R/W：	R/W	R/W	

ビット：	15	8	7	0
	HCCA		–	
初期値：	0		0	
R/W：	R/W		R/W	

▶ HcPeriodCurrentEDレジスタ

ビット：	31	16
	PCED	
初期値：	0	
R/W：	R/W	

ビット：	15	4	3	0
	PCED		–	
初期値：	0		0	
R/W：	R/W		R/W	

▶ HcControlHeadEDレジスタ

ビット：	31	16
	CHED	
初期値：	0	
R/W：	R/W	

ビット：	15	4	3	0
	CHED		–	
初期値：	0		0	
R/W：	R/W		R/W	

〔図2〕USBホストコントローラの各レジスタ構造（一部のレジスタのみ）（つづき）

▶ HcControlCurrentEDレジスタ

ビット：	31〜16
	CCED
初期値：	0
R/W：	R/W

ビット：	15〜4	3〜0
	CCED	−
初期値：	0	0
R/W：	R/W	R/W

▶ HcFmIntervalレジスタ

ビット：	31	30〜16
	FIT	FSMPS
初期値：	0	0
R/W：	R/W	R/W

ビット：	15	14	13〜0													
	−	−	FI													
初期値：	0	0	1	0	1	1	1	0	1	1	0	1	1	1	1	1
R/W：	R/W	R/W	R/W													

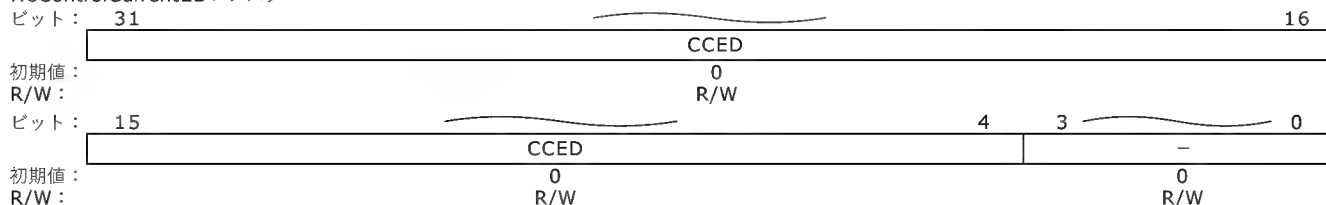
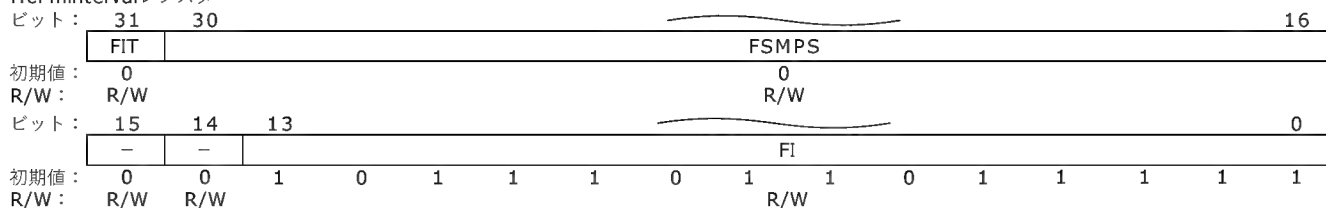
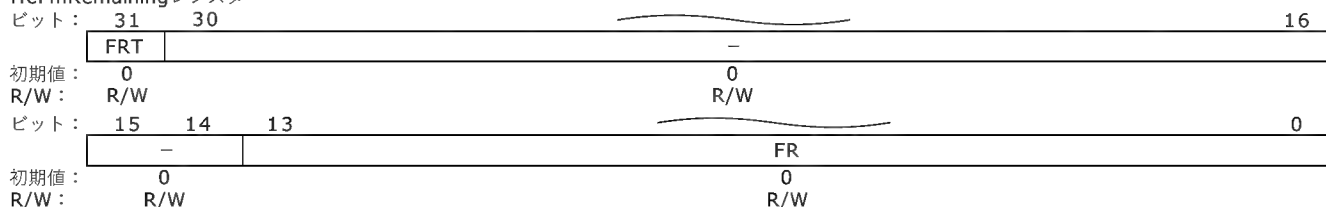
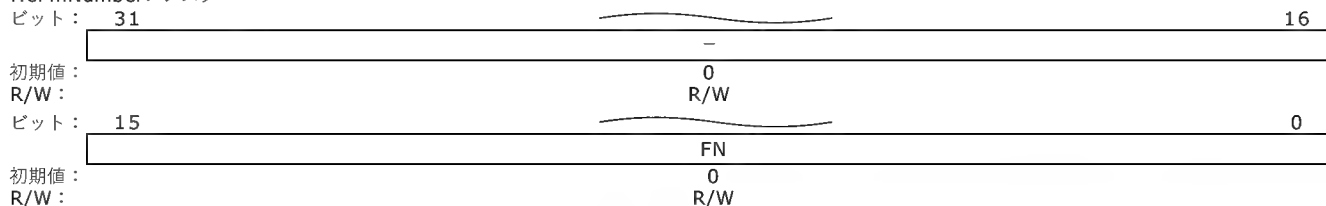
▶ HcFmRemainingレジスタ

ビット：	31	30〜16
	FRT	−
初期値：	0	0
R/W：	R/W	R/W

ビット：	15〜14	13〜0
	−	FR
初期値：	0	0
R/W：	R/W	R/W

▶ HcFmNumberレジスタ

ビット：	31〜16
	−
初期値：	0
R/W：	R/W

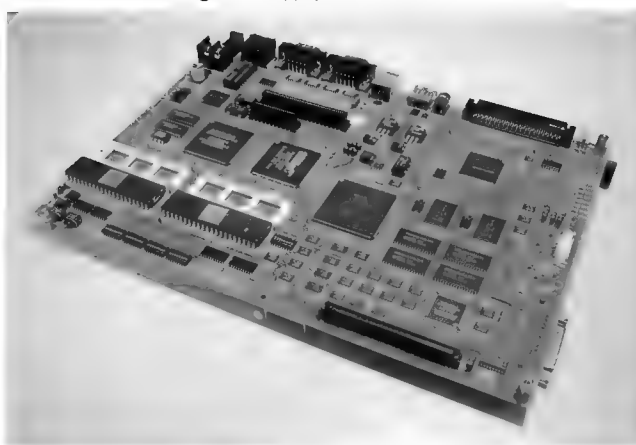
ビット：	15〜0
	FN
初期値：	0
R/W：	R/W

PCMCIA，Ethernet，SuperI/Oなどが搭載されています．また，ROMやRAMの搭載量も，通常の評価には十分な量があります．

今回はUSBホストドライバの評価にのみ使用したので，周辺デバイスのほとんどは未使用でしたが，これだけの周辺デバイスがあれば，かなり面白いアプリケーションが考えられそうです．また機会があれば，他のデバイスもテストしてみたいところです．

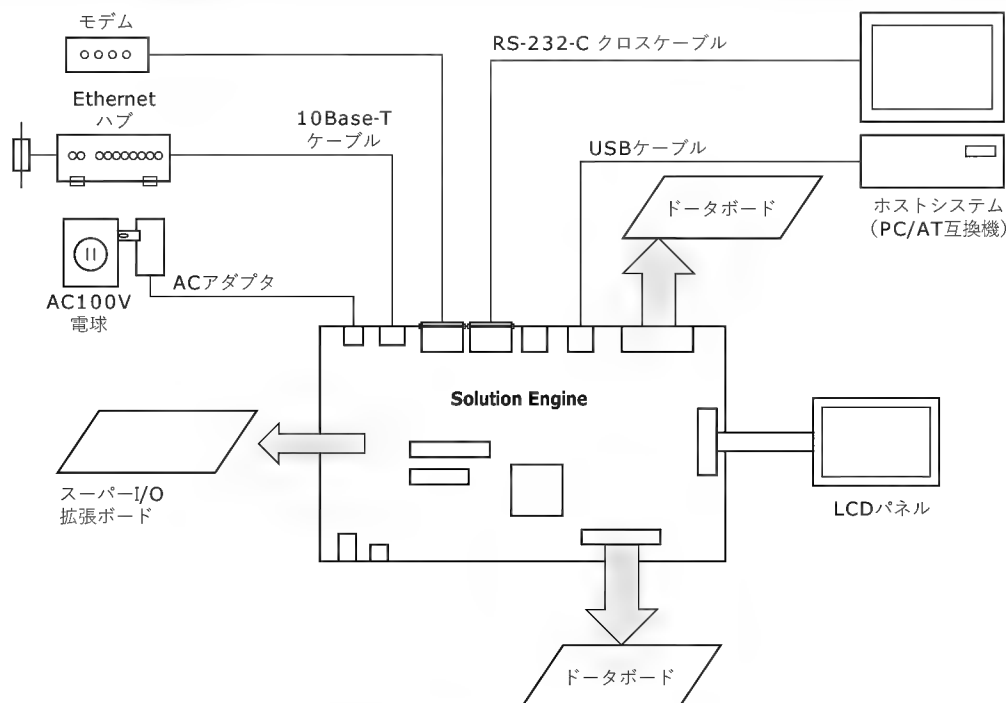
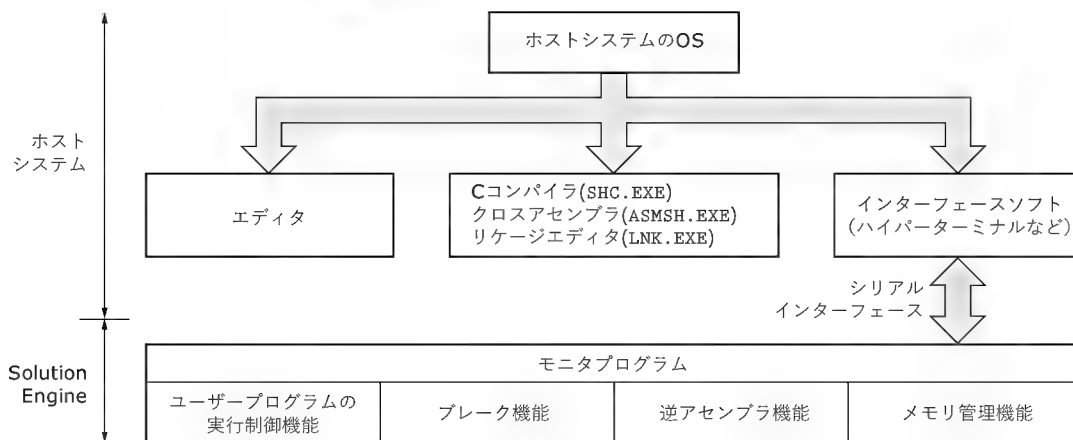
● SolutionEngineの開発環境

次に，SolutionEngineの開発環境について説明します．**図5**(p.89)にSolutionEngine使用時のシステム構成を示します．

ホストシステムとは，RS-232-Cクロスケーブルで接続します．ホストで使用するOSはWindows2000/Me/98などです．その他の周辺装置は，必要に応じて接続します．

図6に，SolutionEngineを使ったソフトウェア開発環境の構成を示します．ソフトウェアは，Windows上のクロス環境で開発ができます．**図6**でホストシステムとなっているのがWindowsになります．開発はWindows上のエディタでコーディングを行い，同じくWindows上のコンパイラでコンパイルをします．コンパイルしたプログラムは，シリアルインターフェースを使ってSolutionEngineにダウンロードされます．SolutionEngineにはモニタプログラムが入っており，ダウンロードしたプログラムの実行やブレーク，ステップ実行などのデバッグ作業を，ホストコンピュータ上で行うことができます．

〔写真1〕SolutionEngine（MS7727SE01）の外観

〔図4〕SolutionEngine のブロック図

〔図5〕SolutionEngineシステム構成図

〔図6〕SolutionEngineを使ったソフトウェア開発環境構成図

● SolutionEngineの対応OS

SolutionEngineは，μITRON，Linux，VxWorksなど，各種のOSに対応しています．今回は，OSにμITRONを採用しました．μITRONは国内では採用実績の多いOSなので，すでに開発経験のある方も多いのではないでしょうか．

今回使用したのは，μITRON4.0仕様に準拠した日立製のリアルタイムOSです．開発環境の画面のようすを**図7**に示します．特徴は次のとおりです．

●業界標準のμITRON仕様に準拠

μITRON4.0仕様に準拠した日立製マイコン用のリアルタイムOS．

●優れたリアルタイム性能

タスク切り替え時間が従来製品よりもさらに高速．

●高機能

HI7000/4シリーズでは，オブジェクトの動的な生成機能など100以上のシステムコールをサポート．多様な応用に対応．

● OS構築パラメータの設定を容易にするコンフィグレータを装備〔**図7(b)**〕

2 USBホストドライバミドルウェアFlexiStackの概要

● FlexiStackとは

FlexiStackは，Phillipsが開発したUSBホスト用のミドルウェアです．FlexiStackは，基本的にはOS非依存で作られてお

〔図7〕日立製 μITRON の開発環境

```
Flexistack - Hitachi Embedded Workshop - [stk_enum.c]
ファイル(F) 編集(E) 表示(V) プロジェクト(P) オプション(O) ビルド(B) ツール(T) ウィンドウ(W) ヘルプ(H)

Flexistack
  Flexistack
    Assembly source file
    Bsd
      stk_enum.c
      stk_hc_dep.c
      stk_kernel.c
      stk_kernellib.c
      stk_lib.c
      stk_os_dep.c
      stk_usbdi.c
    Bsd_inc
    C header file
    C source file
    commoninc
    Hcd
    Hcd_inc
    Dependencies

        stDevRequest.wIndex = 0;
        stDevRequest.wLength = 0;
        uStatus = fnuHcdControlTransfer (uPipeHandle, stDevRequest, NULL);

        if (uStatus)
        {
            fnvDeAssignDevAddr (uHostId, uDevAddr); /* v122 */
            return (uStatus);
        }

        /* Close the pipe handle and reopen the pipe with the device address */
        uStatus = fnuHcdClosePipe (uPipeHandle);

        if (uStatus)
        {
            fnvDeAssignDevAddr (uHostId, uDevAddr); /* v122 */
            return (uStatus);
        }

        /* Save the assigned device address */
        pstHubStruc->typstHubPorts[uPortNum-1].uDevAddr = uDevAddr;

        /* Open the control endpoint pipe again with the device address*/
        uStatus = fnuOpenPipeHc (uHostId, uDevAddr, USB_CONTROL_ENDPOINT_NUM,
                                (uDeviceSpeed ? HCD_LOW_SPEED : HCD_FULL_SPEED), uMaxPacketSize,    /* v123 */
                                HCD_CONTROL_TRANSFER, NULL, NULL, &uPipeHandle, 0);
        if (uStatus)
        {
            fnvDeAssignDevAddr (uHostId, uDevAddr); /* v122 */
            return (uStatus);
        }

        /* Create a device structure and attach it to the device tree */
        if (tybyBuffer[USB_DEV_DESC_CLASS_OFFSET] == USB_HUB_CLASSCODE)

Project Files | Navigation

Building - Flexistack - Debug

Phase SH C/C++ Library Generator starting
Phase SH C/C++ Library Generator finished

Phase SH C/C++ Compiler starting
C:¥HEW2¥Flexistack¥Flexistack¥dbsct.c
C:¥HEW2¥Flexistack¥Flexistack¥sbrk.c
C:¥HEW2¥Flexistack¥Flexistack¥resetprg.c
C:¥HEW2¥Flexistack¥Flexistack¥Flexistack.c
C:¥Mitubishi¥HostStack¥BSD¥stk_usbdi.c
Build | Debug | Find in Files | Version Control
レディ                                Read-write  186/373  38  INS  NUM
```

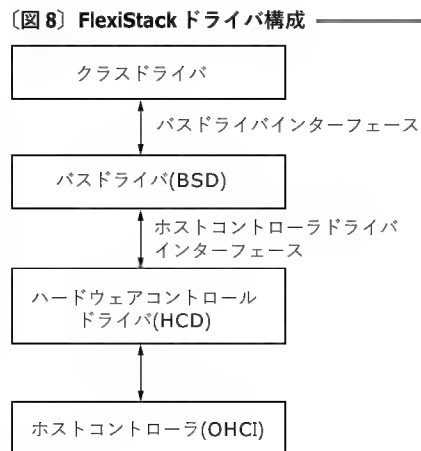
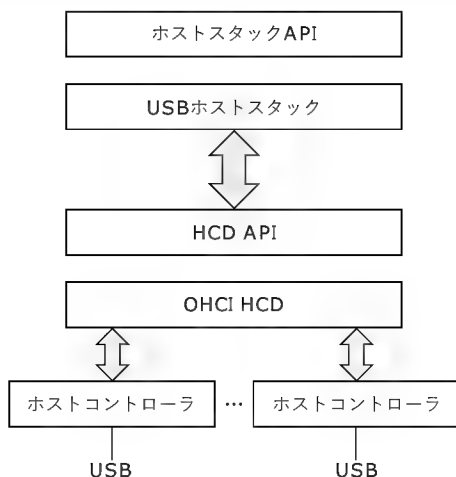
（a）統合開発環境

（b）OS構築パラメータの設定コンフィグレータ

〔図8〕FlexiStack ドライバ構成

クラスドライバ
↕ バスドライバインターフェース
バスドライバ(BSD)
↕ ホストコントローラドライバインターフェース
ハードウェアコントロールドライバ(HCD)
↕
ホストコントローラ(OHCI)

〔図9〕複数のホストコントローラのサポート

〔図10〕RTOS上のFlexiStack論理構成図

り，さまざまなOSへの移植が可能です．実際にμITRONやVxWorks，Linuxといった OSで稼動しています．またCPUについては，x86やSH，MIPSなどのCPUでの動作実績があるようです．

もともとOS非依存で作られているので，これ以外のOSやCPUへの移植も容易にできるようです．ホストコントローラについては，現在のものはOHCIコントローラの対応となっていますが，比較的容易にUHCIコントローラにも対応できるようです．

● FlexiStackの構成

まずFlexiStackの具体的な構成について説明します．**図8**は，FlexiStackのドライバの構成図です．FlexiStackという名前のとおり，全体はスタック構造となっています．図のいちばん下は，ホストコントローラのハードウェアです．ハードウェアを直接操作するのは，HCDと呼ばれるハードウェアコントロールドライバです．ハードウェア依存部分は，すべてこのドライバで吸収されます．このドライバの上に位置するのが，BSDと呼ばれるバスドライバです．バスドライバは，クラスドライバとHCDの橋渡しの役割を果たします．

HCDとBSD間のインターフェースは，ホストコントローラドライバインターフェースと呼ばれるインターフェースで，データのやり取りが行われています．

最上位に位置するのがクラスドライバです．クラスドライバは，各USBデバイスに合わせて用意されます．たとえば，USBマウスにはUSBマウスクラスドライバ，HDDなどにはマスストレージクラスドライバといったドライバが用意されます．クラスドライバとバスドライバの間は，バスドライバインターフェースというインターフェースで通信が行われます．

FlexiStackは，最大で8個までのホストコントローラをサポートしています．複数のホストコントローラのサポートは，HCDで行われるので，これより上位のドライバは，ホストコントローラの数を意識する必要はありません(**図9**)．

● μITRONでのドライバ開発

Windowsのドライバ開発では，アプリケーションに比べると特別なスキルを要求され，また関連する書籍も少ないため，ドライバの開発と聞くと，二の足を踏んでしまう方も多いことでしょう．

μITRONの場合は，Windowsのような保護モードやプラグ＆プレイのための複雑な機構があるわけではありません．基本的にはライブラリ関数の作成と同じなので，とくに難しいことはありません．OSのシステムコールのファンクション部分を作成するようなイメージでよいと思います．ただし，OS側でのドライバ環境のサポートが少ないので，USBホストドライバのように，比較的高機能なドライバを作る場合には，ドライバを作る側に多くの負担がかかります．

USBで実現されるプラグ＆プレイの機構などは，ドライバ側で実現しなければなりません．しかし，今回使用したミドルウェアであるFlexiStackは，プラグ＆プレイの機構をすでにもっているため，この部分を自分で作成する必要はありません．後で詳しく説明しますが，FlexiStackを使用する場合は，OSやCPU依存部の書き直しだけでUSBホストドライバが完成します．

FlexiStackを使用するうえで注意する点は，FlexiStackがタスクを使用している点です．といっても，タスクを使用する部分は，FlexiStack上で関数にまとめられており，さまざまなRTOSに簡単に移植できるようになっているので，とくに問題となるようなことはないでしょう．RTOSに関する詳しい説明はここでは省略しますが，FlexiStackを移植する際に必要となる知識は，きわめて基本的なことだけです．また実際にはこの部分も，今回はμITRON用のFlexiStackを使用したので，とくに意識する必要はありませんでした．

● μITRONでのUSBスタックスタック

それでは実際に，μITRON上にUSBホストスタックを実装

するには，どのような構成になるのかを説明します．**図10**は，RTOS上のFlexiStackの論理構成図です．図でRTOSとなっている部分がμITRONです．ここではアプリケーションからUSBにアクセスする場合の流れを説明します．

アプリケーションがUSBデバイスとデータの通信を行う場合，クラスドライバを呼び出します．クラスドライバの呼び出しは，クラスドライバAPIに基づいて行われます．あるいは，直接クラスドライバを呼び出さず，別のドライバを仲介してクラスドライバの呼び出しを行う場合もあります．たとえば，USBマウスのほかにシリアルマウスもサポートしていて，アプリケーションからは，どちらも同じように扱えるよう，抽象的なマウスドライバが定義されているような場合には，このようなアクセス方法となります．

クラスドライバは，呼び出しが行われるとホストスタックAPIを使い，バスドライバ(BSD)との通信を行います．さらにバスドライバは，ホストコントローラドライバAPIを使ってホストコントローラドライバ(HCD)を呼び出し，そこからホストコントローラとの通信が行われます．

非常にまわりくどいことをやっているように思われるかもしれませんが，複数のホストコントローラをサポートしたり，さまざまな種類のUSBデバイスをサポートするには，このような構成にしたほうが便利です．今回はホストコントローラを実現するため，これらすべてのドライバを操作することになりますが，すでにホストドライバが稼動している場合には，追加するデバイス用のクラスドライバだけ作成すればよいことになります．あるいは，ホストコントローラを1個追加する場合は，HCDのみの変更で済みます．したがって，このようなスタック構成にすることにより，開発者は必要最小限開発工数で済むというメリットがあるのです．

3 FlexiStack移植作業

● 移植手順

FlexiStackは，x86用のμITRON用のソースとして提供されます．実際にはターゲットとなるCPUと，使用するμITRONへの移植作業が必要となります．FlexiStackを移植するにあたっての手順を説明します．移植作業は次の手順で行うとよいでしょう．

(1) 動作環境の明確化
(2) ハードウェア依存部分の対応
(3) RTOS依存部分の対応
(4) 接続デバイスに関する部分の対応
(5) コンパイラに依存する部分の対応
(6) 動作検証とデバッグ

それぞれの手順について詳しく説明します．

(1) 動作環境の明確化

ここではハードウェアとソフトウェアの環境を明確にします．ハードウェアとしては使用するホストコントローラの仕様，割り込み番号，I/Oアドレスやメモリ環境の確認，使用するUSBデバイスなどが挙げられます．またソフトウェアに関しては，使用するRTOS，コンパイラやデバッグ環境，用意するクラスドライバなどを明確にしておきます．

(2) ハードウェア依存部分の対応

FlexiStackでは一部ハードウェア依存の部分があります．この部分はターゲットとなるハードウェアに合わせて書き換えが必要です．移植が必要となる部分は，割り込み処理とI/Oポートの部分です．とくに割り込み関連は，使用するμITRONにより割り込みのサポートがない場合があります．OSのサポートがない場合は，ドライバ側でEOIの処理が必要となります．また，CPU依存の部分も使用するCPUに合わせて修正します．

(3) RTOS依存部分の対応

FlexiStackはμITRON用のソースとして提供されているので，RTOS依存部分はほとんどありませんが，使用するμITRONにより，一部修正が必要になる場合があります．とくに注意が必要な部分は，割り込み処理のサポートの有無やヒープメモリの管理，タスク生成ライブラリ，I/Oアクセスライブラリです．

(4) 接続デバイスに関する部分の対応

使用するホストコントローラやハブの数，インターフェースやエンドポイントの数に合わせて，獲得するメモリサイズを調整します．

(5) コンパイラ依存部分の対応

コンパイラによっては，使用している構造体のデータの中に余分なバイトが付加される場合があります．コンパイルスイッチにより対応できる場合は，コンパイル時にそのスイッチが有効になるように設定します．このようなスイッチがない場合は，プログラム中にデータ転送用バッファを用意し，バイト単位にデータ生成を行う必要があります．

(6) 動作検証とデバッグ

動作検証は，ターゲットとなるUSBデバイスを実際に接続して行います．最初は簡単なデバイスを使ってデバッグするとよいでしょう．たとえばUSBマウスであれば，マウスのスイッチの状態や座標を取得してみるとよいでしょう．今回は，FlexiStackと同様にミドルウェアとして提供されている，マスストレージクラスドライバを使用して，動作検証を行いました．

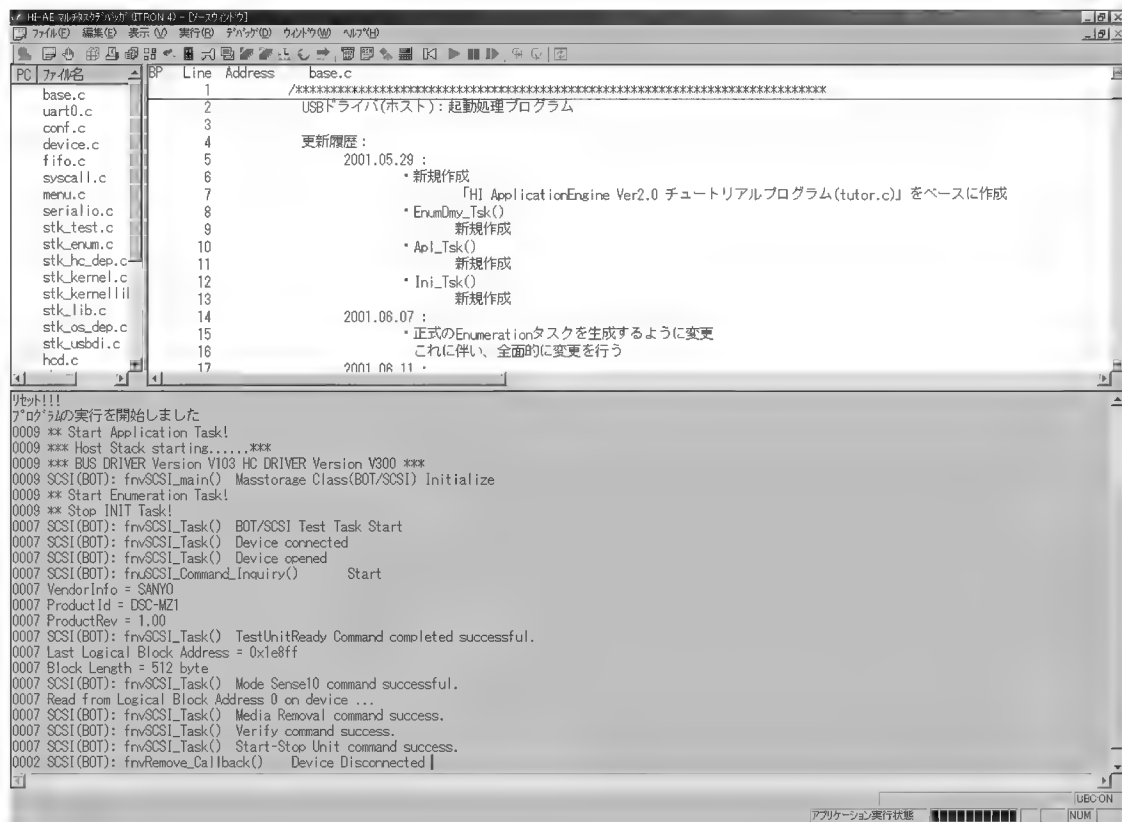
● 移植の実際

実際の移植作業は，提供されるソースコードの必要部分を書き換えていきます．

ソースコードには，ハードウェアやRTOSなどに依存する部分が，次の三つの定数を使用して，条件コンパイルができるようになっているので，この条件コンパイルの部分を必要に応じて書き直していくと良いでしょう．

具体的には次の部分です．

```
#define USBD_HARDWARE_PLATFORM USBD_X86
```

〔図11〕デバッグ中の画面

```
#define USBD_OS_PLATFORM        USBD_UITRON
#define USBD_USB_HC_TYPE        USBD_OHCI
```

定数USBD_HARDWARE_PLATFORMはハードウェアプラットホームに関するものです．FlexiStackはx86用のμITRONのミドルウェアとして提供されます．したがって，これを今回使用するSH7727用に変更する必要があります．具体的には，

```
#define USBD_HARDWARE_PLATFORM USBD_SH7727
```

に書き換え，

```
#if(USBD_HARDWARE_PLATFORM == USBD_X86)
    (中略)
#endif
```

となっているところを，

```
#if(USBD_HARDWARE_PLATFORM == USBD_X86)
    (中略)
#endif
#if(USBD_HARDWARE_PLATFORM == USBD_SH7727)
    SH7727で独自のコード
#endif
```

のように書き換えます．この他，OS_PLATFORMとUSB_HC_TYPEの部分も必要に応じて書き換えますが，今回はOSもホストコントローラの仕様も同じだったため，この部分に関してとくに変更はありませんでした．

最後に，実際に移植のために変更を行ったソースを**リスト1**(次頁)に示します．今回はbase.cファイルとusbhost.cの二つのファイルのみとなりました．

● 動作検証

ソースの修正が終わり，コンパイルが無事通るようになったら，最後に動作検証を行います．実際，何かの製品にFlexiStackを組み込む場合は，その製品で使用するさまざまなUSBデバイス用のクラスドライバを組み込んでテストしなければなりませんが，今回は単なる移植のテストだったので，マスストレージクラスのデバイスでテストを行うことにしました．

マスストレージデバイスとは，USB Implementers Forumで定義されているデバイスクラスで，コンパクトフラッシュやSDカードなどのリーダ/ライタ，あるいはUSB付きのディジタルカメラなど，多くの記憶デバイスが対応しています．最近では，Windows用にドライバ不要のメモリカードリーダが販売されていますが，これはWindowsに最初からマスストレージクラスのドライバがインストールされていて，またこれらのデバイスがマスストレージクラスに準拠しているためです．

今回のテストには，SANYO製デジタルカメラDSC-MZ1を使用しました．テスト用のアプリケーションを作成し，デバッガでデバイス接続時の状態をモニタして，動作を確認しました．**図11**に，エニュメレーション終了後にHostStackより接続を通知

〔リスト1〕実際に移植のために変更を行ったソース

```
/****************************************************************************
    base.cファイル
    USBドライバ(ホスト):起動処理プログラム
****************************************************************************/

/*--------------------------------------------------------------------------/
    初期化ハンドラ
/--------------------------------------------------------------------------*/

void
init_hdr( VP_INT exinf)
{
    T_CTSK          ctsk;                                       /* タスク生成情報 */
    ER_ID           ercd_id;
    void Ini_Tsk( INT stacd);                                   /* 初期化タスク宣言 */

    /* 起動タスク生成 */
    ctsk.tskatr    = TA_HLNG | TA_ACT;                          /* タスク属性 */
    ctsk.exinf= 0;                                              /* 拡張情報 */
    ctsk.task = (FP)Ini_Tsk   ;                                 /* タスク起動アドレス */
    ctsk.itskpri= TSK_PRI_USR_TOP;                              /* タスク起動優先度 */
    ctsk.stksz= 1024;                                           /* タスクスタックサイズ */
    ctsk.stk = NULL;                                            /* スタック領域 */
    ercd_id = acre_tsk( &ctsk);                                 /* タスク生成 */
    if( ercd_id > 0) {
        gTsk_id[ INIT] = ercd_id;
    } else {
        vsys_dwn( 2, ercd_id, 0, 0);                            /* システムダウン(エラー発生時) */
    }
}

/*-------------------------------------------------/
 機 能:起動タスクメイン
 引き数:INT stacd        = 未使用
 戻り値:(void)なし
/-------------------------------------------------*/
#pragma  noregsave(Ini_Tsk)
void
Ini_Tsk( INT stacd)
{
      ER           ercd;
      T_CTSK       ctsk;
      ULONG        ulRet;
      Serial_Open();                                            /* シリアル入出力初期化処理 */
      UsbHard_Init();                                           /* USB関連ハードウェア初期化 */
      /* Enumerationタスク生成 */
      ulRet = fnuStartUsbHostStack();
      if( ulRet != 0 ) goto EXIT_TASK;                          /* エラーチェック */
      else gTsk_id[ENUM] = UITRON_ENUM_TASK_ID;
      mon_smn01_sndmsg(26, "** Start Enumeration Task!");       /* モニタにメッセージ出力 */
EXIT_TASK:                                                      /* タスク終了 */
    mon_smn01_sndmsg( 18, "** Stop INIT Task!");                /* モニタにメッセージ出力 */
    exd_tsk( );
                                                                /* 自タスク終了&削除 */
}

/*-------------------------------------------------/
 機 能:USB関連ハードウェア初期化処理
 引き数:なし
 戻り値:(void)なし
 特 記:USBHの初期化は,USB HostStack側で行う.
     USBFは未使用のためリセットをかけたままにする.
/-------------------------------------------------*/
void     UsbHard_Init(void)
{
    UCHAR           uchTmp;
    USHORT          ushTmp;

    IPRG = 0x0000;                                              /* USBHの割り込み禁止(同時にUSBF0,USBF1,AFEIFも禁止:暫定) */
    SRSTR |= (SRSTR_USBH|SRSTR_USBF);                           /* USBH,USBFをリセット */

    EXPFC = (unsigned short)0x0000u;                            /* USBホストを使用 */
    EXCPGCR = (unsigned char)0x30u;                             /* USB用48MHzクロック設定 */
    PDCR &= (unsigned short)0xcfffu;                            /* PTD6ポートを「その他機能」(ULCK入力)に設定 */
    SRSTR &= (~SRSTR_USBH);                                     /* USBHのリセットを解除 */

    return;
}
```

(a) base.c

Information ── アイエニウェア・ソリューションズ・インク,日本法人を設立
アイエニウェア・ソリューションズ・インクは,モバイルおよびワイヤレス事業における日本市場展開のために,日本法人アイエニウェア・ソリューションズ(株)を設立した.

〔リスト1〕実際に移植のために変更を行ったソース(つづき)

```
/****************************************************************************
    usbhost.c ファイル
    host controller 初期化処理等
****************************************************************************/

/*------------------------------------------------------------------------/
    x86用の fnuHostEnumerate()をベースにして、全面的に修正
    Host Controller Enumeration 処理
/------------------------------------------------------------------------*/
ULONG fnuHostEnumerate(void)
{
    ULONG     uHostTotal;                                 /* Number of host found */
    ULONG     host idx;
    ULONG     uHostType;
    st HOST NODE* pstHostNode;
    uHostType = HOST TYPE;                                /* OHCI or UHCI, defined in host conf.h */

    /* Initialize the host table */
    for (host idx = 0; host idx < MAX HOST+1; host idx++) {
        tpstHostTable Hcd[host idx] = NULL;
    } /* for */

    uHostTotal = 0;

    pstHostNode = AllocHostNode();
    if (pstHostNode != NULL) {
        uHostTotal++;
        pstHostNode->uHostBase          = USBH BASE ADR;      /* Base Address */
        pstHostNode->uHostType          = uHostType;          /* Host Type */
        pstHostNode->uHostId            = uHostTotal          /* HCD's Host ID number */
        pstHostNode->uVendorId          = 0;                  /* Vendor ID */
        pstHostNode->uDeviceId          = 0;                  /* Device ID */
        pstHostNode->uIntLevel          = 0;                  /* Host controller's IRQ line */
        tpstHostTable Hcd[uHostTotal] = pstHostNode;
    } /* if */

    return uHostTotal;

} /* fnuHostEnumerate() */

/*----------------------------------------------------------------------------------------/
    x86用の fnvBspHookIsr()をベースにして, 全面的に修正(暫定対応として、処理をすべて削除)
    割り込みコントローラ設定
    割り込みハンドラ登録
/----------------------------------------------------------------------------------------*/
void UsbIntHndr(void)
{
    extern    ULONG UsbHostIsr(ULONG uHostId);
    ULONG uHostId;

    uHostId = 1;
    UsbHostIsr(uHostId);

    return;
}

/***********************************************************************
void fnvBspHookIsr(st HOST NODE *pstHostNode)
Input:
    pstHostNode
        Pointer to the host node data structure.

Output:
    None

This function is called from the HCD API fnuHcdHostInit() to install
a USB host controller interrupt service routine and enable the
interrupt of the system's interrupt controller.

This function is hardware-dependent and should be rewritten to fit
the particular hardware of the platform.
***********************************************************************/
void fnvBspHookIsr(st HOST NODE *pstHostNode)
{
#define hi sr dsp              0x00001000
#define INTLVL USBH            9                          /* USBH割り込みレベル */
#define INTC USBH              0xa00                      /* USBHの例外コード */

    T DINH    pk dinh;                                    /* 割り込みハンドラ定義情報 */
    ER ID     ercd id;                                    /* リターンパラメータ */
```

(b) usbhost.c

〔リスト1〕実際に移植のために変更を行ったソース（つづき）

```
    pk dinh.inhatr = TA HLNG;                                    /* ハンドラ属性(高級言語) */
    pk_dinh.inthdr = (FP)UsbIntHndr;                             /* ハンドラアドレス */
    pk dinh.inhsr = (0x40000000|(INTLVL USBH<<4)|hi sr dsp);

    /* 起動時のSR */
    ercd id = def inh(INTC USBH, &pk dinh);
    if (ercd id!=E OK) {                                         /* 属性不正 or パラメータエラー */
        return;
    }

/* To Do : 割り込み禁止をかけてから設定する */
    IPRG = ((INTLVL_USBH << 12) & 0xf000);
/* USBHの割り込み許可(同時にUSBF0,USBF1,AFEIFを禁止) */

    return;
} /* End of fnvBspHookIsr() */
```

（b）usbhost.c（つづき）

され，SCSIコマンドを処理しているところをトレース中のデバッガの画面を示します．

まとめ

駆け足で説明したため，十分な説明になっていなかったかもしれませんが，移植までの流れは大体わかってもらえるのではないかと思います．

FlexiStackは，移植性を非常によく考えて作られており，実際，移植のために書き換えたコードは，今回掲載したリスト程度の簡単なものでした．ただ，この手の移植作業は，移植のためのコーディングよりも，ターゲットデバイスのハードウェアの理解，ターゲットOSやコンパイラの理解，あるいは開発環境の習得など，本質以外の部分にかかる労力のほうが大きくなります．

今回は，まったく新しい環境で開発を行ったので，これらの部分の習得に時間がかかりましたが，よく慣れた環境であれば，本質的なコーディングの部分だけに注力できるので，かなり簡単に移植できるのではないかと思います．ただ，やはり移植作業を行うためには，FlexiStackそのものの習得も不可欠なので，必要最低限の知識を得る労力が必要になります．各種のCPUや各種のRTOSに対応するコードを作るのはたいへんだと思いますが，もう少し修正が必要な部分が，簡潔にまとめられているとありがたかったと思います．実際に変更する部分は，確かに少ないのですが，変更が必要な個所を探すのに，エディタなどで，

```
#if(USBD_HARDWARE_PLATFORM == USBD_XXXX)
```

といった行を検索していって，修正が必要かどうか判断しながら修正を加えていくというのは，かなり根気のいる作業です．今後の改善を望みます．メーカー側でも今後も著名なRTOSやCPUに対応していくようなので，すでに対応済みのCPUやRTOSであれば，移植作業はほとんど必要ないのかもしれません．

■問い合わせ先

● FlexiStack
(株)スティル
TEL：03-5785-1775
URL：http://www.stil.co.jp
E-mail：sales@stil.co.jp

● SH7727
(株)日立製作所　半導体グループ
〒100-0004　東京都千代田区大手町二丁目6番2号(日本ビル)
URL：http://www.hitachisemiconductor.com/jp/
E-mail：(半導体カスタマサービスセンタ)csc@sic.hitachi.co.jp

● Solution Engine(MS7727SE01)
(株)日立超LSIシステムズ 営業部
〒185-0014　東京都国分寺市東恋ヶ窪3-1-1
TEL：042-359-2210(代表)　FAX：042-359-2213
URL：http://www.hitachi-ul.co.jp/
SuperH Solution Engine技術サポートセンター
E-mail：sh-sengn@hitachi-ul.co.jp
TEL：0120-60-1213

せりい・しげき　(株)ソリトンウェーブ

On-The-Go対応USBコントローラとプロトコルスタック

芹井滋喜

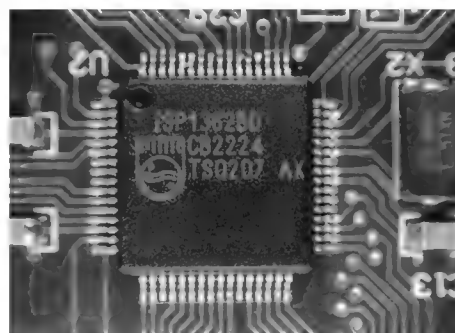
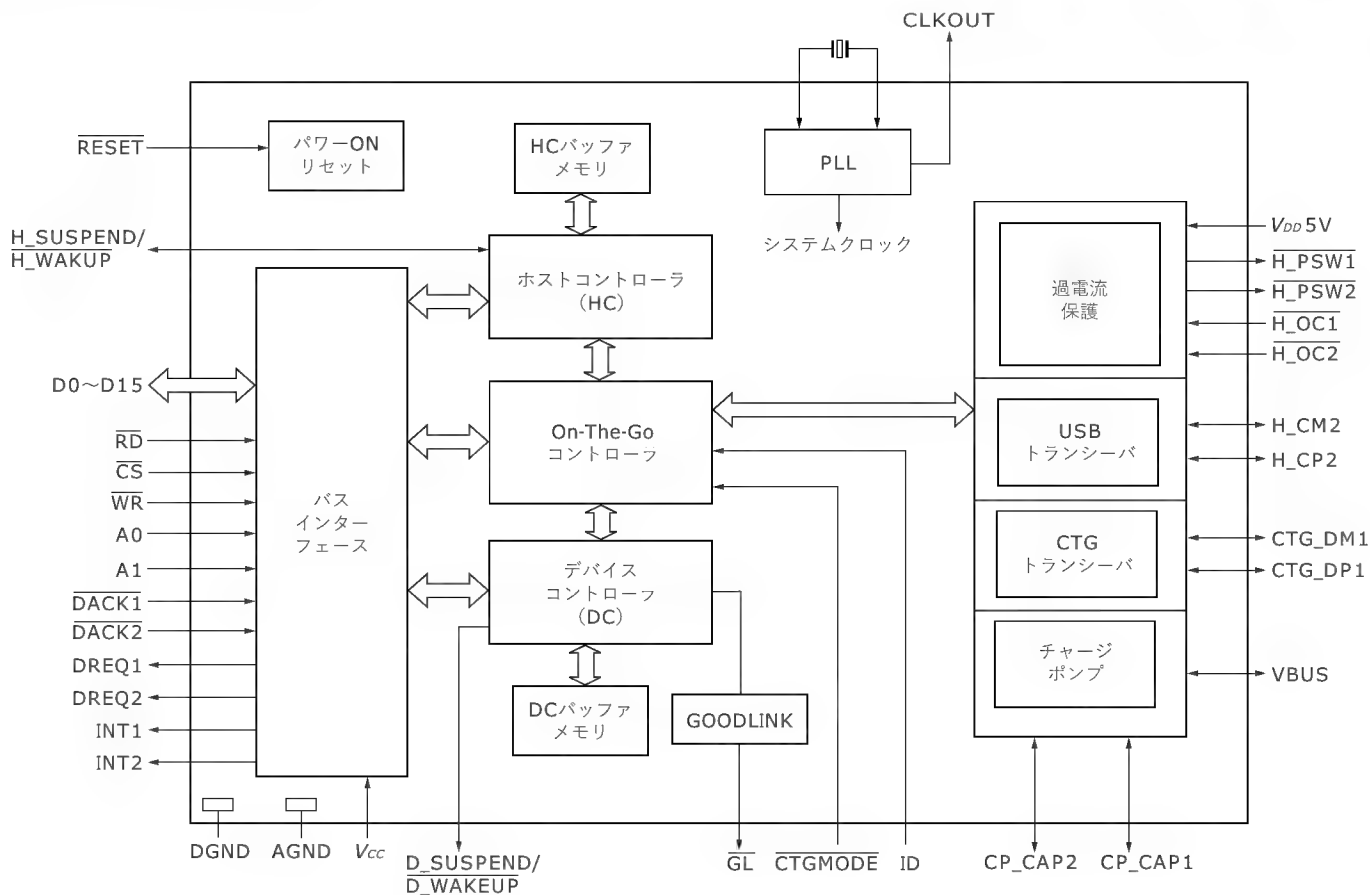
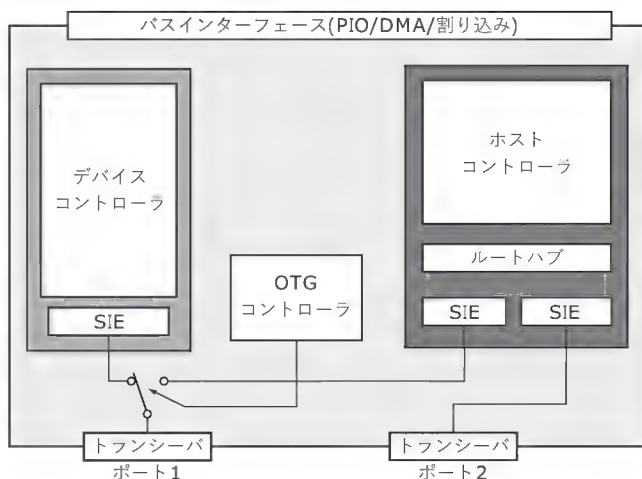
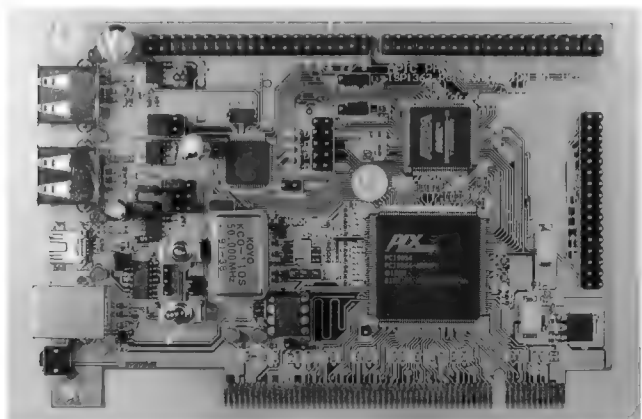
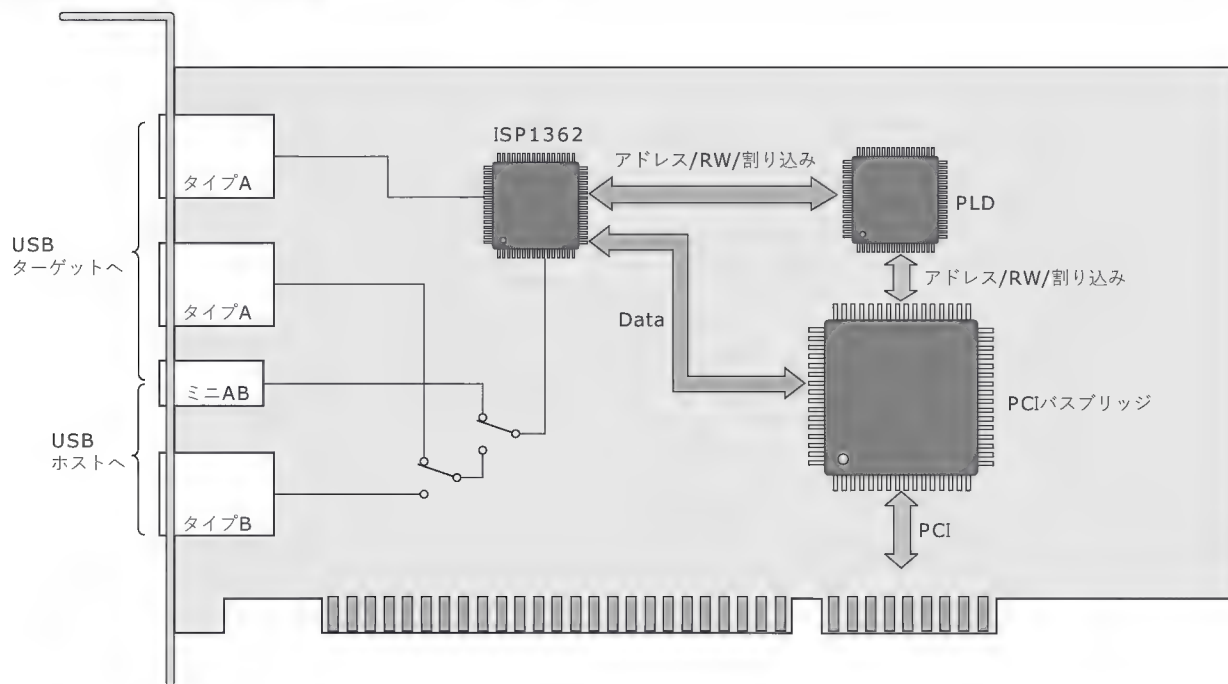
● USB On-The-Goとは

従来のUSB機器は，PC周辺機器としての接続を想定し，システムを構築していました．しかし近年，民生分野では，パソコンを介すことなくUSB機器間をつなぐニーズが増えています．USB On-The-Go (以下OTGと略)は，USB搭載機器間でのデータ交換を可能にする新しい規格です．このOTG規格の登場で，USB対応機器の新しい製品カテゴリとして，1台の機器でUSBのホストとしての役割と，USB周辺機器(デバイス)としての役割の，両方の機能を兼ね備えた製品が可能になります．

これにより，たとえばPDAやデジカメを直接ストレージ機器やプリンタにつなぎ，操作することが可能になります．二つの機器を接続する場合，各機器に接続するケーブルのコネクタによって，それぞれの役割が決まります．OTGの場合は，新しいコネクタの規格であるミニABコネクタを使用します．ただしこの場合，ソフトとしてOTGのプロトコルを使ってデータ交換を行います．基本的に，ファイル転送を開始した機器がホストになります．エンドユーザーはホストとデバイスを意識する必要なく機器間を接続できます．

● OTG対応のハードウェア

現在，数社の半導体メーカーからOTG対応チップが供給されています．ここでは，ISP1362(フィリップス)を例に説明します．**図A**にISP1362のブロック図を，**写真A**に外観を示します．

〔写真A〕ISP1362の外観

〔図A〕ISP1362のブロック図

〔図B〕両コントローラとポートの構成

〔写真B〕ISP1362評価ボードの外観

OTG対応チップは次のような特徴があります．

- OTGコントローラ内蔵
- ホストコントローラ(HC)およびデバイスコントローラ(DC)を独立して内蔵

OTGペリフェラル動作時でも，ホストコントローラ(HC)側でポート2(ホスト専用ポート)につながれたデバイスの制御が継続可能になります．

- ホストコントローラ(HC)とポート1/2の間はルートハブを介して接続
- 汎用CPUバスインターフェース

また，ホストコントローラとデバイスコントローラおよび各ポートの構成関係を**図B**に示します．

● ISP1362評価ボード

ISP1362には，PCIに対応した評価ボードがあり，パソコンのマザーボードに評価ボードを差し込んで簡単に評価できる環境が整っています．**写真B**にISP1362評価ボードの外観を，**図C**に評価ボードのブロック図を示します．

写真左側の上二つのコネクタはタイプAコネクタで，ホスト側になります．ルートハブを内蔵しているので2ポートあるわけです．いちばん下はタイプBコネクタでデバイス側になります．そしてタイプAとタイプBの間にある小さいコネクタが，ミニABコネクタです．

● OTGに必要なソフトウェア

OTGを実現するには，次のソフトウェアが必要になります．

- 周辺機器側(デバイス側)ファームウェア

従来のデバイス側のファームウェア

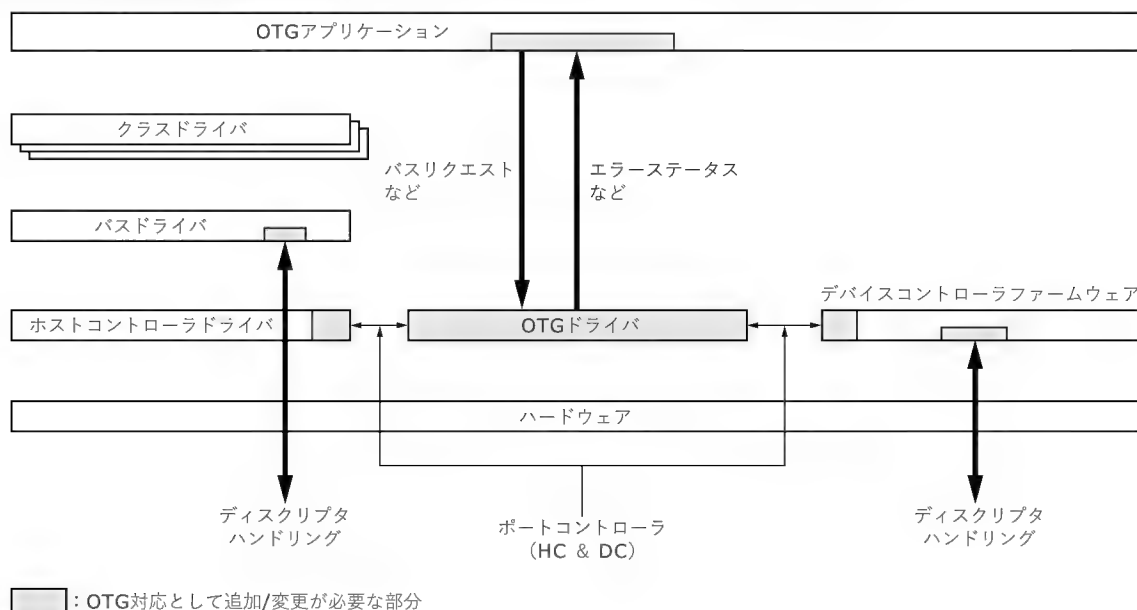
- ホスト機能実現のためのホストスタック

ホストコントローラドライバ(HCD)，バスドライバ(BD)，クラスドライバ

- OTGドライバ

OTGプロトコルの実現

〔図C〕ISP1362評価ボードのブロック図

〔図D〕OTG対応として追加/変更が必要な部分

●OTGアプリケーション

BusRequestの発生，NoSilentFailureの扱いなど

また，クラスドライバには，機器の目的に応じて各種のクラスドライバが必要になります．HID，マスストレージクラス，プリンタクラス，コミュニケーションクラス，オーディオクラスなどがUSB規格として規定されています．OTG対応の特徴としては，パソコンと接続をする場合と異なり，接続機器を限定することが可能なため，機器メーカー独自のクラスドライバを搭載する場合もあります．

図Dは，OTG非対応のプロトコルスタックをOTG対応に変更する場合の例です．

なお，第3章で紹介したFlexiStackには，OTG対応版も発売されています．**図E**にOTG対応版FlexiStackのブロックダイアグラムを示します．このOTG対応版FlexiStackは，各種RTOSとCPUに対応をしており，最近ではPDAやプリンタなどに使われているようです．

コラム SH7727でいくか，ISP1362でいくか

第3章で解説しているSH7727は，1チップにOHCI準拠のUSBホストと，USBターゲットコントローラの両方が内蔵されています．つまり，OTG対応のISP1362を使わなくても，USBホストにもターゲットにもなり得る機器を実現できるわけです．

OTG対応デバイスを使う場合と，SH7727のようにUSBホストとターゲットを両方内蔵したデバイスを使う場合を，さまざまな点から比較してみましょう．

▶ハイスピードモードを使いたい場合

現状のOTGは，480Mbpsのハイスピードモードと1.5Mbpsのロースピードモードはオプションとなります．また，現在発売されているOTGデバイスは，12Mbpsのフルスピードモードに対応したものだけです．したがって，現状でハイスピードで動作するデバイスを実現する場合は，EHCI(ハイスピード対応ホストコントローラ)＋ターゲットという構成をとる必要があります．

▶ミドルウェアと開発コスト

OTGは比較的新しい規格なので，従来構成のほうが，ミドルウェアなどのライブラリが入手しやすいという利点もあります．またOTGの場合，マスタ/スレーブの切り替えのためのプロトコルが新たに追加されているので，開発コストはOTGのほうが高くなるでしょう．

▶使い勝手はOTGが上

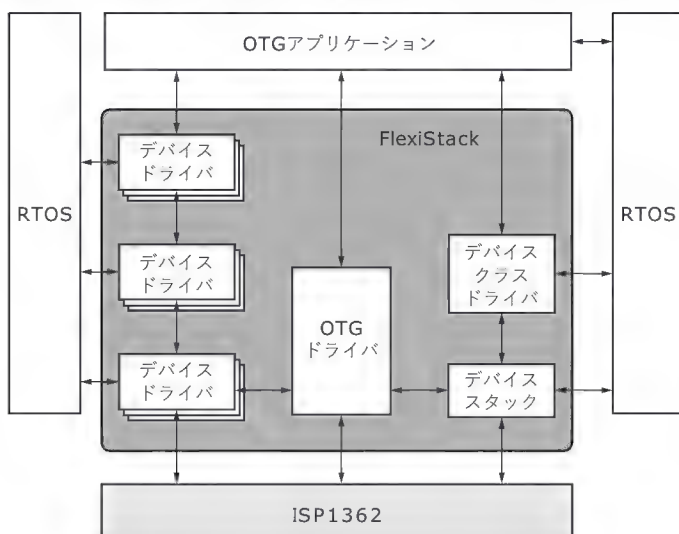
逆にOTGの利点として，ユーザー側の混乱が少ないという点があります．従来のように，必ずパソコンが介在するのではなく，OTG機器同士であれば，何も考えずに接続することができます．従来であれば，USBディジタルカメラとUSBプリンタをもっていても，間にパソコンがなければ画像を印刷できませんでした．

OTG対応のディジタルカメラであれば，パソコンなしでも印刷ができます．

また，ミニABコネクタは，ミニAコネクタもミニBコネクタも接続できるので，ユーザーがホスト機器とクライアント機器を意識する必要はありません(デフォルトでは，Aコネクタを接続した側がホストになるが，Host Negotiation Protocolによりケーブルを入れ換えずにホストとクライアントの機能を切り替えることができる)．

▶コネクタのスペースファクタ

さらにコネクタも，A/B兼用のミニABコネクタ一つですむので，携帯電話などのように，実装スペースが限られる用途には，OTGが非常に適しているでしょう．

〔図E〕OTG対応ソフトウェアのブロック図

● OTGの課題

今後は，多くの半導体メーカーからUSB OTGチップが販売されると思われ，OTG対応機器開発がしやすくなると考えられますが，問題はそれだけではなく，下位のコミュニケーション層に対応した下位レベルのドライバの搭載が必要になります．これは次のような状況の場合に，やっかいな問題となります．

OTG対応のディジタルカメラの例を考えます．このディジタルカメラを，OTG対応のプリンタに接続して画像を直接印刷できるのがOTGの利点です．ところがOTGでは，あらかじめ周辺機器のドライバをもっていなければなりません．そこであらゆるOTGプリンタで写真を印刷できるようにするためには，ディジタルカメラが使用するすべてのOTGプリンタに対応したドライバを備えていなければ，写真を印刷できないという，現実的に実現困難な問題が発生します．

ディジタルカメラの場合は，最近発表されたDPS（Direct Printer Service）規格がユーザーにとっては大きなメリットとなりそうです．今後，この統一されたインターフェースを利用した製品が増えてくることが予想されます．

また，現在はOTGではありませんが，パソコンを使わずにディジタルカメラのデータを保存できるUSBホストを搭載しているMOが販売されています．MOがUSBホストで立ち上がっているときはディジタルカメラをUSBデバイスとしてMO側にデータを転送し，画像データが保存されます．また，そのデータを編集する際には，MOがデバイスになり，PCやゲーム機がUSBホストとなって，MOから画像データを吸い上げて編集ができるような製品です．使用範囲は限定されますが，一般的にはMOをホストに使いたいという用途は限られているので，現実的な解決策の一つといえるかもしれません．

Windowsなどのいわゆるパソコン製品であれば，新しい周辺機器にも，ドライバのインストールで簡単に対応できますが，組み込み機器の場合は，あらかじめドライバを用意しなければなりません．まったく新しい周辺機器であれば，まだあきらめがつくかもしれませんが，上記の例のように，新しいプリンタを買ったら印刷できなくなってしまうようでは，あまり便利な機器とはいえません．こういった問題は，インターフェースの標準化を推し進めることにより，解決されていくことでしょう．

■問い合わせ先

● FlexiStack
(株)スティル
TEL：03-5785-1775
URL：http://www.stil.co.jp/
E-mail：sales@stil.co.jp

● ISP1362
日本フィリップス(株)
TEL：03-3740-5130
URL：http://www.semiconductors.philips.com/buses/usb/products/otg/isp136x/
E-mail：sc-toiawase.japan@philips.com

せりい・しげき　(株)ソリトンウェーブ

USB2.0で追加された新しいプロトコル

最新USBハブチップにみるトランザクショントランスレータの動作

山下泰弘

USB2.0では480Mbpsという高速な転送速度を活かすために，ダウンストリームポートにフルスピード/ロースピードどちらのデバイスが接続されても，アップストリーム側へはハイスピードのパケットに変換する，トランザクショントランスレータ機能が規定された．ここではトランザクショントランスレータの動作と，最新ハブチップにおけるマルチトランザクショントランスレータの利点などについて解説する．

(編集部)

はじめに

USB2.0が正式に公開されてから，約2年弱が経過しました．USB2.0ホストコントローラカードをはじめとして，ハードディスクドライブ，CD-RWドライブ，またビデオキャプチャユニットなど，たくさんの製品が市場で見られるようになりました．

今回はその中でも，単なるポートを増やすための機器として見られがちなUSBハブに焦点を当て，とくにUSB2.0で追加された新たなプロトコルやアーキテクチャについて述べます．

1 USBハブの概要

● USBハブの内部構成

USBハブのメインとなるUSBハブコントローラICは，大きくはハブリピータとハブコントローラの二つに分けられます(**図1**)．

(1) ハブリピータ

ハブリピータは，アップストリーム-ダウンストリーム間のUSB接続の確立と切断を行います．また，デバイス着脱の検知や通信異常の検知なども行います．

(2) ハブコントローラ

ハブコントローラは，ホスト-ハブ間のUSB通信に必要なプロトコルを処理し，ハブ自身や，そのダウンストリームポートのステータスの管理，またステータスのレポートを行います．

USBホストコントローラは，ハブコントローラに対してHub Class Specific Requestを送ってハブを制御し，ハブリピータへUSB通信を流すことで全体の通信を行います．

● 市販されるUSBハブ

USBハブには，USB2.0対応品(最高480Mbps)，USB1.1対応品(最高12Mbps)という区別以外に，いくつかの機能上の違いがあります．

(1) セルフパワード方式/バスパワード方式

USBハブに限りませんが，装置としてACアダプタなどの自己供給電力を必要とするものと，それを必要とせずUSBバス電源のみで動作するものにわけられます．

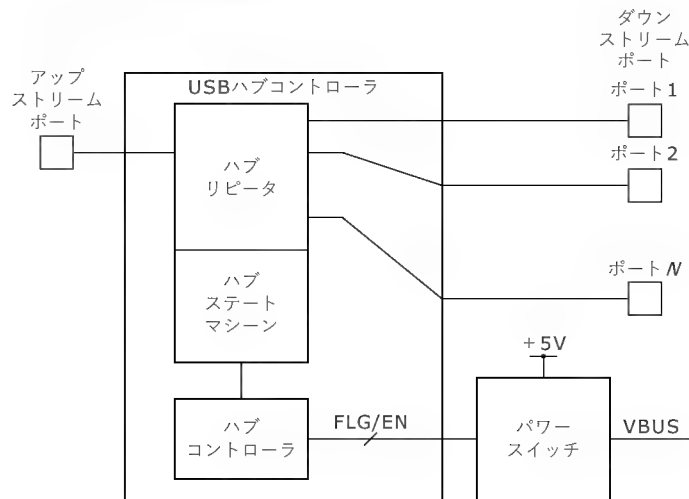
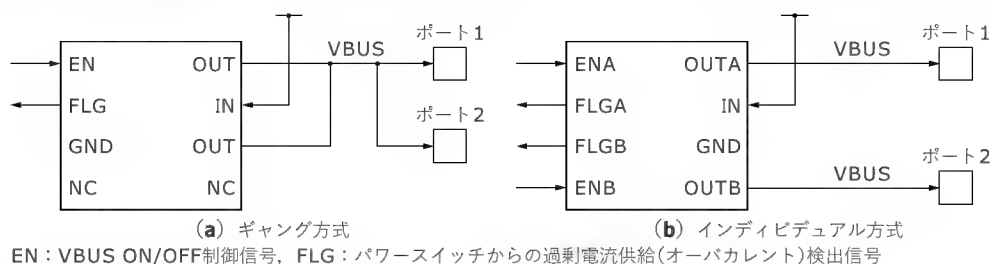
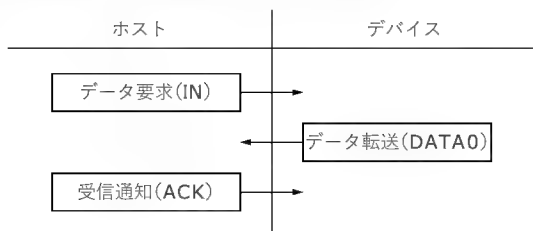
バスパワード方式では，自己供給電力は必要ないのですが，ダウンストリームポートへ供給できる電力が，5V/500mAから5V/100mAに変更されます．そのため，オプティカルマウスやフラッシュカードリーダなど，バスパワーで動作するもので比較的消費電力の大きなUSBデバイスを接続する際には注意が必要です．

(2) ギャング(Gang)方式/インディビデュアル(Individual)方式

前述のとおり，USBハブではダウンストリームポートに対し5V/500mA，または5V/100mAの電力が供給できます．しかしながら，それよりも大きな電力を消費するUSBデバイスがポートに接続された場合は，そのデバイスをUSBトポロジから切り離さなければいけません．USB規格では，バスパワードハブの

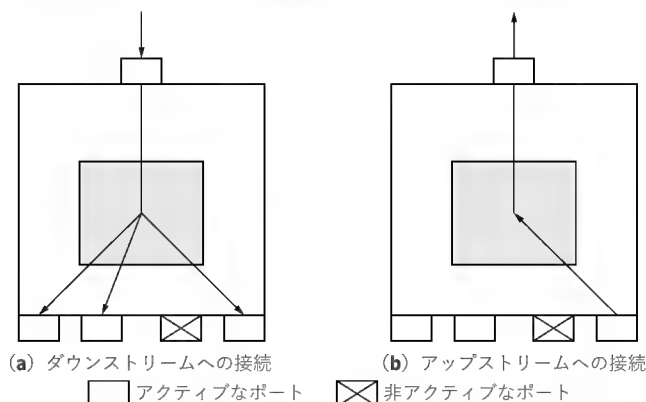
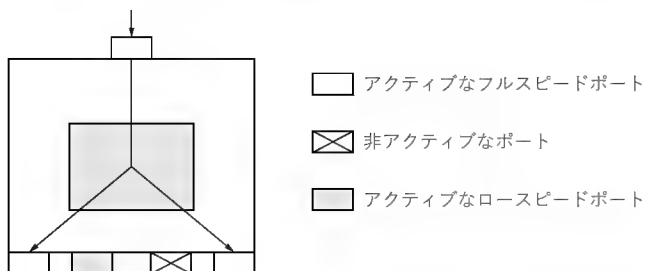
〔図1〕USBハブの内部構成

〔図2〕VBUS制御用パワースイッチの例

(a) ギャング方式　(b) インディビデュアル方式
EN：VBUS ON/OFF制御信号，FLG：パワースイッチからの過剰電流供給(オーバカレント)検出信号

〔図3〕データ受信におけるトランザクション

場合はパワースイッチを必須に，セルフパワードハブでは許可としています．このパワースイッチ制御をUSBダウンストリームポートごとに行う方式をインディビデュアル方式，ダウンストリームポート全体に対して行う方式をギャング方式といいます(**図2**)．

ギャング方式では，ハブコントローラのピン数を削減することができ，コストを安くできる反面，電流の過剰供給が発生した際には，すべてのダウンストリームポートの電流が切られます．

〔図4〕ダウンストリームへの接続とアップストリームへの接続

(a) ダウンストリームへの接続　(b) アップストリームへの接続
□ アクティブなポート　☒ 非アクティブなポート

〔図5〕ロースピードポートが混在したハブでのフルスピード通信

(3) シングルトランザクショントランスレータ方式/マルチトランザクショントランスレータ方式(USB2.0対応ハブのみ)

トランザクショントランスレータは，USB2.0になって新たに追加された，USBハイスピードハブに必要とされる機能です．詳しくは後述しますが，内蔵されているトランザクショントランスレータが一つである場合と，複数である場合では，そのハブに接続されるフルスピード/ロースピード製品のパフォーマンスに違いが現れます．

2 USB1.1のプロトコル

● USB通信の流れ

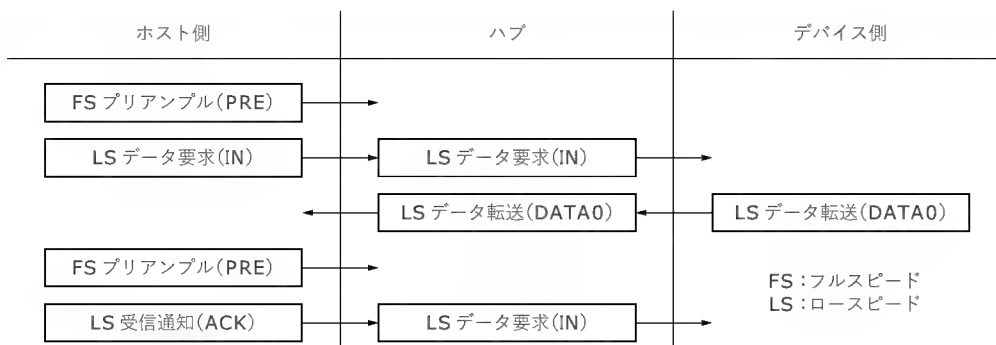
USB通信はトランザクションという転送単位からなり，そのトランザクションはおもにトークンパケット，データパケット，ハンドシェイクパケットの3種類のパケットからなり立っています．すべてのトランザクションはホストからのトークンパケットでスタートし，デバイスが通信のマスタとなることありません．**図3**は，USBデバイスからUSBホストへデータを受信する際の一連のトランザクションです．

この例では，ホストからのデータ要求トークン(IN)に対し，デバイスが送信するデータをもっているため，データ転送(DATA0)を行っています．送られたデータはホストによって正しく受信され，受信通知によりハンドシェイクを行っています．これら一連のパケットのやりとりによりトランザクションがなり立っています．

● フルスピードUSBハブを介したUSB通信

USBハブには，前述のとおりリピータが内蔵されており，接続が確立されたダウンストリームポートとアップストリームポート間で信号を伝送します．これにはいくつかの決まりがあります．

(1) 接続が確立されたすべてのダウンストリームポートに，アップストリームからのトランザクションをブロードキャストする(**図4**)．
(2) ただし，ダウンストリームポートに接続されたデバイスがロースピードである場合，フルスピード信号をそのロースピードポートに流してはいけない(**図5**)．
(3) ホストは，USBハブに接続されたロースピードデバイスと通信する際，プリアンブル(PRE)パケットを各パケットに

〔図6〕
USBハブを介したデータ受信におけるロースピードトランザクション

付随させ，ハブはそのPREパケットを検知することでロースピードポートと通信を行う(**図6**).

このようにフルスピードUSBハブを介した通信では，フルスピード/ロースピードのビットタイムをもったUSB信号がアップストリーム-ハブ間を流れます．ロースピードの遅い通信はバス上のボトルネックを生みますが，フルスピードとロースピードのビットレートが8倍と比較的小さいことから，パフォーマンスにそれほど大きな影響は与えません．

3 USB2.0のプロトコル——トランザクショントランスレータ

● ハイスピードUSBハブを介したUSB通信

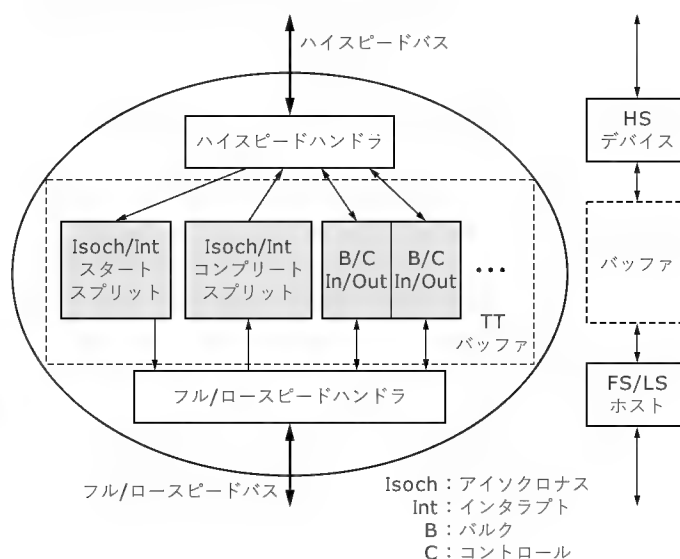
前述のフルスピードUSBハブと比較し，ハイスピードUSBハブにおいては，ハイスピードとフルスピードのビットレートに40倍もの大きな差があるため，それらの遅いビットタイムをもった信号がバスを共有すると，非常に大きなボトルネックになり得ます．そのためUSB2.0では，トランザクショントランスレータという新たな仕様が導入されました．

トランザクショントランスレータは，アップストリームに対してはハイスピードUSBデバイスとしてふるまい，ダウンストリームに対してはフル/ロースピードUSBホストとしてふるまうことで，アップストリーム-USBハブ間の通信を，すべてハイスピード信号で取り扱うことができるというものです．

トランザクショントランスレータの内部ブロックは**図7**のようになっています．ハイスピード信号化されたフル/ロースピードトランザクションは，後述するスプリットトランザクションのマナーにしたがってバッファリングされ，ダウンストリームポートに接続された実際のフル/ロースピードデバイスの信号へ変換されます．USB2.0では，遅延の許されないアイソクロナス/インタラプト通信用のPeriodic Bufferと，そうでないバルク/コントロール通信用のNon-periodic Bufferをもつことを定めています．

● スプリットトランザクション

同じハイスピード信号で伝播するフル/ロースピードのトランザクションを，フル/ロースピードとして切り分けるために，USB2.0ではスプリットトランザクションという機構が追加されました．スプリットトランザクションは，スタートスプリットと

〔図7〕**トランザクショントランスレータの概要**

コンプリートスプリットという二つのプロセスによりなり立っています．**図8**にスプリットトランザクションの例を示します．

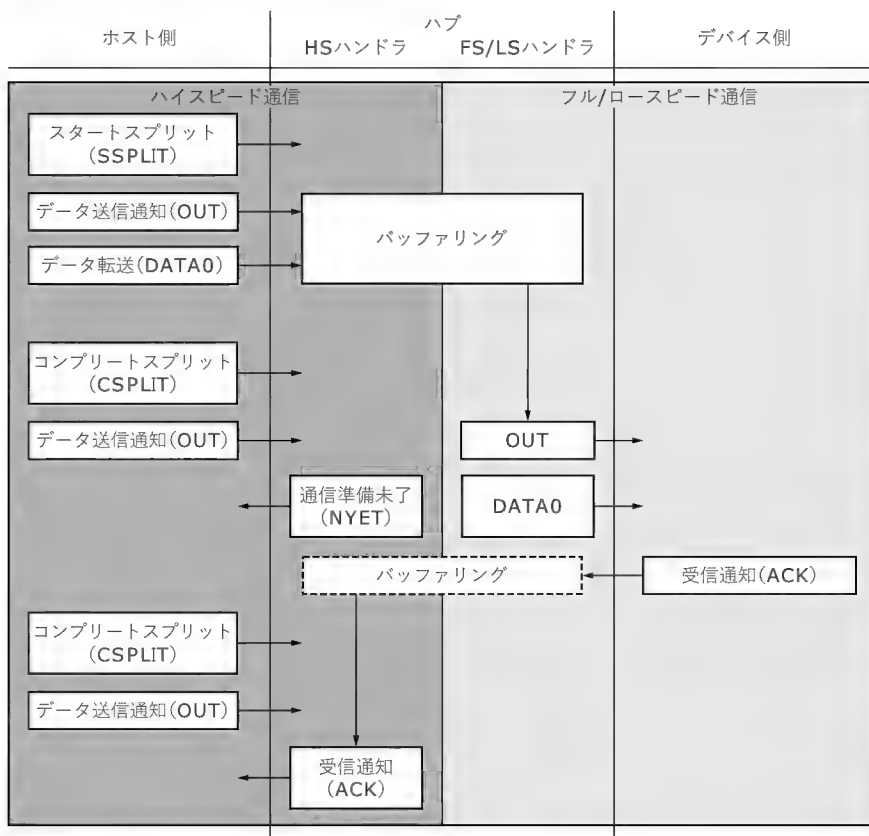
スタートスプリットのプロセスにおいて，SSPLITというUSB2.0にて新たに追加されたパケットをハブに送信します．その後，フル/ロースピード通信をハイスピード信号で送信します．この時点では，フル/ロースピード通信の内容はトランザクショントランスレータ内のバッファに格納され，フル/ロースピードハンドラより通信が解決(コンプリート)するまで保持されます．

コンプリートスプリットのプロセスにおいて，CSPLITという新たに追加されたパケットをハブに送信します．この時点において，SSPLITにて発行したフル/ロースピード通信に対するレスポンスがバッファ内に格納されていれば，それをアップストリームへレスポンスし，未解決であればNYET(Not Yet)を返します．

このように，アップストリーム-ハブ間に遅い信号を伝送させないことで，USBバス上にフル/ロースピードが混在してもボトルネックが全体へ波及しないようにしています．

● シングルトランザクショントランスレータ

さて，トランザクショントランスレータについては説明しまし

〔図8〕スプリットトランザクションの例

たが，各社のハイスピードUSBハブコントローラによって，その実装方法が異なります．それは，ハイスピードハブコントローラに一つだけトランザクショントランスレータを導入する方法と，複数個を導入する方法です．まずは，トランザクショントランスレータが一つのみである場合について説明します．

トランザクショントランスレータは，そのハブに接続されたフル/ロースピードデバイスに対しトランザクションが発行されるたびに使用されます．そのため，シングルトランザクショントランスレータを導入しているハイスピードUSBハブでは，同時に二つ以上のフル/ロースピードトランザクションを処理することができません．そのため，**図9**のようにそのハブのダウンストリームポートに複数のフルスピード，ロースピードデバイスを接続した場合，通信のボトルネックが生じます．

● マルチトランザクショントランスレータ

マルチトランザクショントランスレータ

コラム1 マルチトランザクショントランスレータ方式を採用したハイスピードUSB 4ポートハブコントローラ TetraHub

サイプレスのCY7C65640(呼称：TetraHub)は，世界初のマルチトランザクショントランスレータ方式を採用したハイスピードUSB×4ポートハブコントローラIC(**写真A**)です．ハイスピードUSBロゴの取得は2002年8月に完了しており，現在量産出荷中となっています．

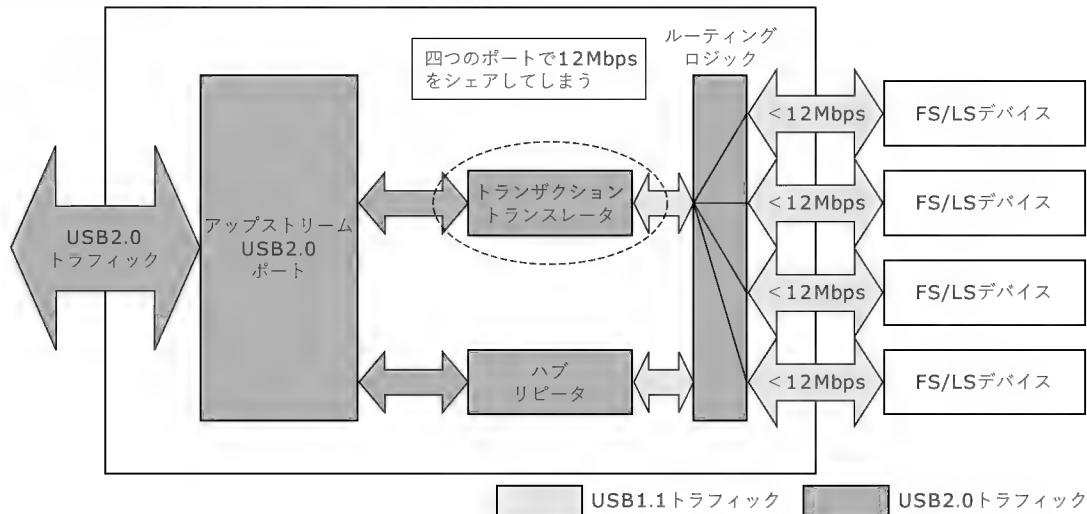
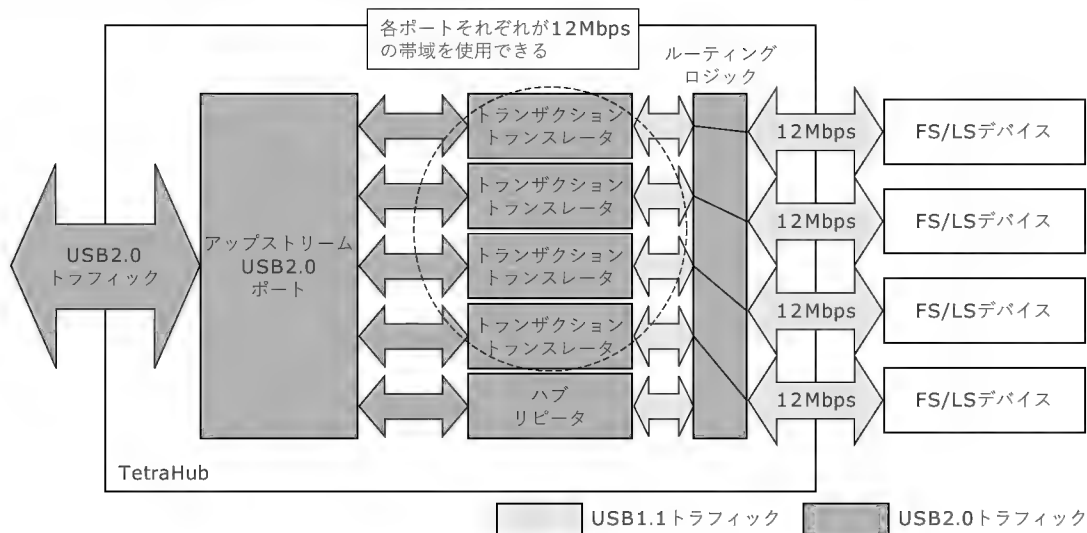
新しく"Tetra"アーキテクチャが採用されているTetraHubは，4本のダウンストリーム周辺機器ポート用に四つのトランザクショントランスレータが搭載されています．これにより，これまでのトランザクショントランスレータが一つしか使用されない，現在普及しているハブよりも優れた性能を発揮します．

CY7C65640の特徴を示します．

- USB2.0ハブ 4ダウンストリームポート
- VID，PID，Non-Removableポートなど，ユーザーアプリケーション仕様に応じたさまざまな設定オプション
- 小型56ピンQFNパッケージ(8mm×8mm)
- エニュメレーション用プルアップ，プルダウンレジスタをデバイスに内蔵

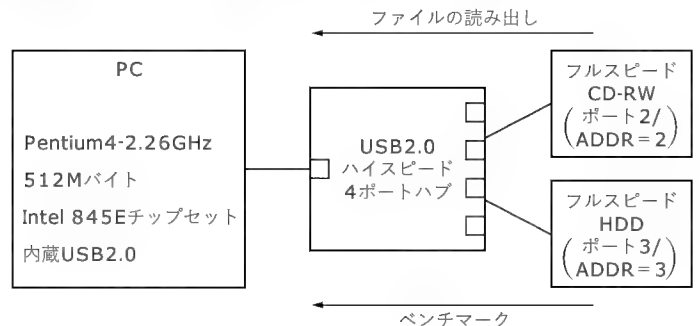
トランザクショントランスレータは，USB 1.1と2.0の間の上位互換性と下位互換性を確保するうえで非常に重要です．TetraHubデバイスは，ファームウェアの介入を必要としない固定機能のソリューションであるため，設計上のリスクが低減され，開発時間が短縮されます．独立型ハブ，モニタハブ，ポートリプリケータ，およびドッキングステーションなどにおいて高帯域幅コントローラとして使用することができます．また，TetraHubコントローラを使用すると，新たなホストコントローラを追加することなく，PC上のUSBポート数を増やすことができます．TetraHubリファレンス デザインキットCY4602も現在提供中です．

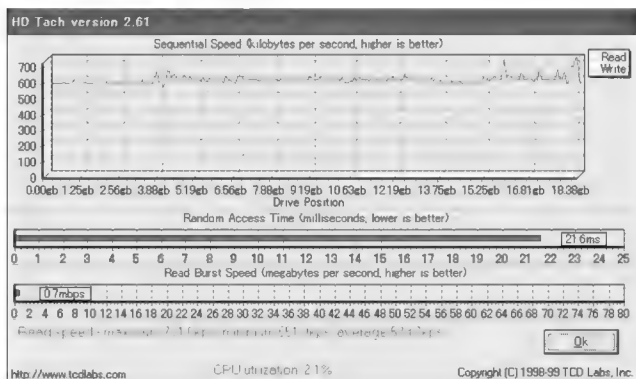
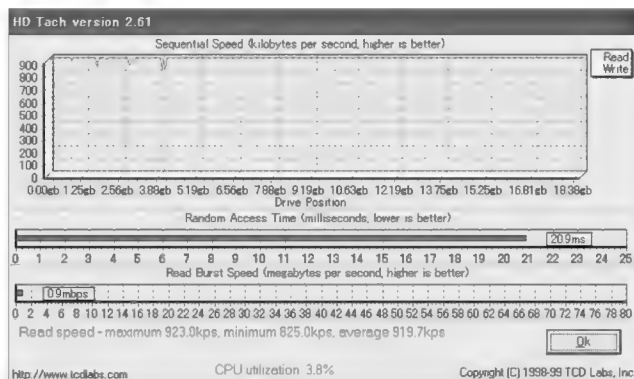
〔写真A〕CY7C65640(TetraHub)の外観

〔図9〕シングルトランザクショントランスレータでのFS/LS通信

〔図10〕マルチトランザクショントランスレータでのFS/LS通信

は，シングルトランザクショントランスレータにあるような，複数個のフル/ロースピード接続におけるパフォーマンス低下を解消するために利用される方式です．具体的な導入例としては，**図10**のようにダウンストリームポートごとにトランザクショントランスレータを導入することです．

ポートごとにトランザクショントランスレータを導入することで，ポートごとに一つずつプロセスできるようになります．その結果，ポートごとに12Mbpsのフルレートを使用できるようになります．

● シングルトランザクショントランスレータとマルチトランザクショントランスレータの比較

前述したシングルトランザクショントランスレータと，マルチトランザクショントランスレータのハブを用いて，実際に比較評価を行いました．テストセットアップとして，**図11**のような環境を用意し，CD-RWドライブよりファイルを定期的に読み出しつつ，ハードディスクの読み出し速度を測定しました．テスト用ハブとしては，シングルトランザクショントランスレータハブとしてD720110AGC(NEC)を搭載したハブと，マルチトランザクショントランスレータハブとしてCY7C65640(サイプレス)を

〔図11〕パフォーマンステストベンチ

〔図12〕パフォーマンス測定

(a) シングルトランザクショントランスレータ

(b) マルチトランザクショントランスレータ

搭載したハブを用意しました．テスト結果を**図12**に示します．

結果からもわかるとおり，二つ以上のフル/ロースピードデバイスを接続した環境下では，トランザクショントランスレータの個数はそのデバイスのパフォーマンスに大きく影響を与えます．

● バスアクティビティの比較

実際のバスアクティビティを比較してみます．**図13**の(**a**)と(**b**)を比較してわかるとおり，シングルトランザクショントラン

コラム2 なぜ現在のUSB2.0は480Mbpsの実力が出ないのか？

● USBストレージのパフォーマンス

すでに市場には記憶装置を中心としてUSB2.0製品があふれており，USB1.1製品との価格差もほとんどなくなりました．

そのなかで，「USB2.0製品は480Mbpsには遠く及ばないパフォーマンスしか出せていない」という雑誌などのコラムや，市場の声を最近よく耳にします．また最近流通しはじめたインテル製のUSB2.0ホストコントローラのほうがパフォーマンスが良いという話も聞きます．じつは，これには理由があります．ここで，その理由が何なのかを検証してみます．

テストには，USB2.0対応ハードディスクドライブを使用しました．HDDドライブにはIBM製IC35L040AVVA07-0（UltraDMA-5対応），USB2.0-ATA変換ICにはサイプレス（旧In-System Design）のISD-300A1を使っています．テストでは比較のため，USBを介さないPC内蔵のIDEポート，インテル製ICH4，そして現在PCIアドオンカードでもっとも広く使用されているNEC製μD720100AGMの3種類の環境を用意しました．

結果を**表A**に示します．これからもわかるとおり，かなりパフォーマンスに差があります．

● USBアナライザを使ってデータ転送の状況を見る

まずDisk Write時を中心に，USBバスアクティビティをトレースしてみることにしました．

図Aおよび**図B**のトレースの結果からわかるように，インテル製ホストでは1マイクロフレーム間に8個，NEC製ホストでは1マイクロフレーム間に5個しかOUTトランザクションが発行されていないことがわかります．この原因は，各トランザクションの間に，インテルでは15μs弱，NECでは約21μsのインターバルがあるためです．

同じようにリード時についてもトレースしたところ，INトランザクションの数はインテル製で約10個（インターバルは12μs弱），NEC製で約6個（インターバルは約20μs）という結果が得られました．

この結果よりまず，データ転送レートとしての最大帯域が，イン

〔表A〕パフォーマンス測定結果（単位：Kバイト/秒）

	リード			ライト		
	平均	最大	最小	平均	最大	最小
内蔵IDE	36720	47228	16738	23926	31122	23926
インテル（USB）	27033	28493	18924	18751	20744	15557
NEC（USB）	17906	18293	10975	13246	14060	11737

〔図A〕ライト時のバスアクティビティ（インテル製チップ）

スレータの場合，スタートスプリットからコンプリートスプリットまでにより長い時間を必要とします．この理由は，ダウンストリームポートに対するフルスピードトランザクションが同時に複数プロセスできないため，バッファリングされたフルスピードトランザクションがなかなか解決できないためです．

次にハブ-HDD間のバスアクティビティを比較します．**図14**の(**a**)と(**b**)ではより顕著な違いが見られます．シングルトランザクショントランスレータでは，トランザクショントランスレータを複数のデバイスで共有しているため，ハブ-HDD間の通信にも，ハブよりブロードキャストされたCD-RWに対するハンドシェイクパケットが見られます．

マルチトランザクショントランスレータでは，それぞれのポートに独立したトランザクショントランスレータをもつため，他のポートに対するバスアクティビティによりバス帯域を阻害されることはありません．**図9**，**図10**のブロック図にあるトランザクショントランスレータを，USBホストとして置き換えればわかりやすいかと思います．

まとめ

ふだんなにげなく使っているUSBハブでも，いくつかのバリエーションがあることはわかっていただけたかと思います．しかし，ここに記した仕様のすべてが実際の製品の仕様欄に記載されていることはあまりありません．パフォーマンスを活かすためには，これらの仕様を確認したうえで，システムに合致したハブを探すことが必要です．

やました・やすひろ　日本サイプレス(株)　応用技術グループ

テル製ホストの場合はリードで40Mバイト/秒，ライトで32Mバイト/秒，NEC製ホストの場合はリードで24Mバイト/秒，ライトで20Mバイト/秒に限定されることがわかります．USB通信は，常にホスト側より生成されるトークンパケットにより通信がスタートするため，トランザクション間のインターバルはパフォーマンスに大きな影響を与えます．

さらにトレースを続けたところ，もう一つBulk-Only-Transportにおけるボトルネックが見つかりました．Bulk-Only-Transportでは，装置へコマンドを発行するためのCBW(Command Block Wrapper)と，デバイスよりステータスを取得するためのCSW(Command Status Wrapper)によりデータはラッピングされますが，Windows標準ドライバを使う限りにおいては，CBW-DATA間，DATA-CSW間，そしてCSW-CBW間の遷移に1マイクロフレームを要してしまっています．これらのオーバヘッドにより，最大帯域よりもさらに低いパフォーマンスしか実現できなくなっています(**図C**)．

● 今後の高速化に期待

今回のテスト結果を見るかぎり，とくにNEC製のホストコントローラ使用時において，トランザクション間のホスト側のインターバルタイムが長いことが大きな課題であるようです．また，Windows標準ドライバにもパフォーマンス改善の余地がありそうです．

NECがより高速化したホストコントローラICの出荷を予定していることからもわかるとおり，USB2.0ホストコントローラは，各社ともまだ第1世代の段階であり，それほど遠くない未来に480Mbpsに近いパフォーマンスを実現できることが期待されています．

〔図B〕ライト時のバスアクティビティ(NEC製チップ)

〔図C〕CBW-DATA-CSW通信

〔図13〕ホスト-ハブ間のバスアクティビティ

Transaction	Split	Hub Addr	Port	IN	ADDR	ENDP	Handshake / Data	Time
36	SSplit Bulk	1	3	0x96	3	1	ACK 0x4B	2 767 µs
37	CSplit Bulk	1	2	0x96	2	2	NYET 0x69	7 867 µs
38	CSplit Bulk	1	3	0x96	3	1	NYET 0x69	2 633 µs
39	CSplit Bulk	1	2	0x96	2	2	NYET 0x69	7 800 µs
40	CSplit Bulk	1	3	0x96	3	1	NYET 0x69	2 633 µs
41	CSplit Bulk	1	2	0x96	2	2	NYET 0x69	7 800 µs
42	CSplit Bulk	1	3	0x96	3	1	NYET 0x69	2 667 µs
43	CSplit Bulk	1	2	0x96	2	2	T 0 Data 64 bytes	7 900 µs
44	CSplit Bulk	1	3	0x96	3	1	NYET 0x69	2 633 µs
45	SSplit Bulk	1	2	0x96	2	2	ACK 0x4B	12 467 µs
Packet 139	SOF 0xA5	Frame # 1918 ?	CRC5 0x10	Pkt Len 14	Time 366 ns	Time Stamp 00000 0386 3339		
46	CSplit Bulk	1	3	0x96	3	1	NYET 0x69	1 767 µs
47	CSplit Bulk	1	2	0x96	2	2	NYET 0x69	8 8[illegible] µs
48	CSplit Bulk	1	3	0x96	3	1	NYET 0x69	2 733 µs
49	CSplit Bulk	1	2	0x96	2	2	NYET 0x69	7 867 µs
50	CSplit Bulk	1	3	0x96	3	1	NYET 0x69	2 700 µs
51	CSplit Bulk	1	2	0x96	2	2	NYET 0x69	7 800 µs
52	CSplit Bulk	1	3	0x96	3	1	T 0 Data 64 bytes	2 867 µs
53	CSplit Bulk	1	2	0x96	2	2	NYET 0x69	7 833 µs
54	SSplit Bulk	1	3	0x96	3	1	ACK 0x4B	2 633 µs
55	CSplit Bulk	1	2	0x96	2	2	NYET 0x69	7 667 µs
56	CSplit Bulk	1	3	0x96	3	1	NYET 0x69	2 700 µs
57	CSplit Bulk	1	2	0x96	2	2	NYET 0x69	7 667 µs
58	CSplit Bulk	1	3	0x96	3	1	NYET 0x69	2 733 µs

HDDに対するスタートスプリット（Transaction 36, 54）
HDDに対するコンプリートスプリット（Transaction 52）

(a) シングルトランザクショントランスレータ

Transaction	Split	Hub Addr	Port	IN	ADDR	ENDP	Handshake / Data	Time
9	SSplit Bulk	1	3	0x96	3	1	ACK 0x4B	2 567 µs
10	CSplit Bulk	1	2	0x96	2	2	NYET 0x69	7 700 µs
11	CSplit Bulk	1	3	0x96	3	1	NYET 0x69	2 633 µs
12	CSplit Bulk	1	2	0x96	2	2	T 0 Data 64 bytes	7 933 µs
13	CSplit Bulk	1	3	0x96	3	1	NYET 0x69	2 600 µs
14	CSplit Bulk	1	2	0x96	2	2	ACK 0x4B	7 733 µs
15	CSplit Bulk	1	3	0x96	3	1	NYET 0x69	2 600 µs
16	CSplit Bulk	1	2	0x96	2	2	NYET 0x69	7 733 µs
17	CSplit Bulk	1	3	0x96	3	1	NYET 0x69	2 633 µs
18	CSplit Bulk	1	2	0x96	2	2	NYET 0x69	7 700 µs
19	CSplit Bulk	1	3	0x96	3	1	T 1 Data 64 bytes	2 900 µs
20	CSplit Bulk	1	2	0x96	2	2	0x69	7 767 µs
21	SSplit Bulk	1	3	0x96	3	1	ACK 0x4B	2 567 µs
22	CSplit Bulk	1	2	0x96	2	2	NYET 0x69	7 767 µs
23	CSplit Bulk	1	3	0x96	3	1	NYET 0x69	2 633 µs
24	CSplit Bulk	1	2	0x96	2	2	T 1 Data 64 bytes	7 967 µs
25	CSplit Bulk	1	3	0x96	3	1	NYET 0x69	2 600 µs
26	SSplit Bulk	1	2	0x96	2	2	ACK 0x4B	7 800 µs
27	CSplit Bulk	1	3	0x96	3	1	0x69	2 600 µs
28	CSplit Bulk	1	2	0x96	2	2	NYET 0x69	6 800 µs
Packet 89	SOF 0xA5	Frame # 1269 ?	CRC5 0x0F	Pkt Len 12	Time 966 ns	Time Stamp 00000 0401 4653		
29	CSplit Bulk	1	3	0x96	3	1	NYET 0x69	2 700 µs
30	CSplit Bulk	1	2	0x96	2	2	NYET 0x69	8 200 µs
31	CSplit Bulk	1	3	0x96	3	1	T 0 Data 64 bytes	2 900 µs

HDDに対するスタートスプリット（Transaction 9, 21）
HDDに対するコンプリートスプリット（Transaction 19, 31）

(b) マルチトランザクショントランスレータ

〔図14〕ハブ-HDD間のバスアクティビティ

(a) シングルトランザクショントランスレータ

(b) マルチトランザクショントランスレータ

USB機器開発における USBアナライザの活用法

谷本和俊

USBアナライザには，数十万円台のものから数百万円を超えるものまで，さまざまな価格帯のものが存在する．また，ロジアナやオシロスコープから発展してきたものや，ICEやエミュレータから派生してきたもの，さらにはキャプチャだけでなくパケットジェネレート機能をもつものまでさまざまである．ここでは，それぞれにどのような特徴があり，どのように使い分ければよいかなど，USBアナライザの活用法について解説する．

（編集部）

本章では，富士通デバイス社製USB2.0アナライザ“USB ZERONE”を使ったUSB機器開発について説明しますが，アナライザ紹介にありがちな機能の羅列ではなく，USBアナライザ全般の解説と，どのような機能をどのような場面で使うのかといった点などを紹介します．他社製アナライザを使っている読者にも，アナライザ活用の一助になると考えています．

また，高価なアナライザほど多機能で，使いこなすまでに時間がかかると思われている読者も多いことでしょう．さらに，USB機器の開発が本格化し，これからアナライザを導入しようとする読者もおられるでしょう．そのような読者に，スムーズなアナライザ導入のために活用していただければ幸いです．

なお，混同されがちな用語“キャプチャ”，“トレース”について，本稿では明確に区別して使用します．“キャプチャ”とは，USBバス上に実際に流れるデータを単純に記録すること，“トレース”とは，キャプチャしたデータをアナライザの機能を使って解析することと定義します(国内ではキャプチャのことをトレース，トレースのことを解析と表現する場合もある)．

1 USBアナライザの種類

USBアナライザの導入にあたっては，現状では多種多様な選択肢が用意されています．価格もさまざまであり，またそれぞれのアナライザが得意とする分野・目的も異なります．まずはじめに，これらのアナライザの種類を分類しておきます．

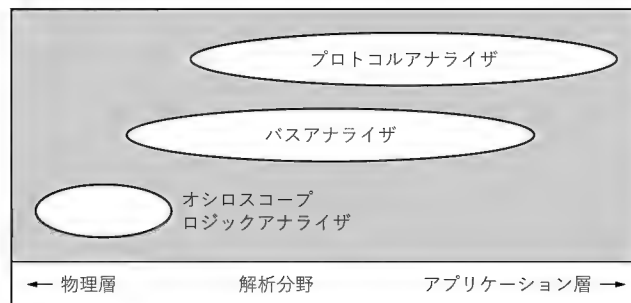
アナライザにはそれぞれ得意とする分野があります．大まかに分類すると，物理層よりのほうが得意かアプリケーション層よりのほうが得意かという違いです．これはアナライザメーカーによってほぼ決まるといえます．ロジックアナライザやオシロスコープなど，物理層の事象解析を行う測定器の発想で開発されたアナライザは，文字どおり物理層よりの解析が得意なアナライザに仕上がっています．ハードウェア開発者やLSI設計者は，この種のアナライザのほうが感覚的に理解しやすいでしょう．

それに対して，インサーキットエミュレータやソフトウェア開発ツールなどの発想から生まれたアナライザは，アプリケーション層の解析が得意なアナライザです．こちらは，アプリケーションソフトウェア，組み込みソフトウェアの開発者にとって親しみやすいものでしょう．

物理層よりの解析が得意なアナライザを「バスアナライザ」，アプリケーション層よりの解析が得意なアナライザを「プロトコルアナライザ」と区別することもあります．さらに，解析機能を省いたキャプチャ機能だけのものをバス(ライン)モニタと呼ぶこともあります．

それぞれ一長一短があるのですが，お互いの良いところを取り込みながら機能アップをしてきており，現在ではあまり違いがなくなってきています．**図1**に，ロジックアナライザやオシロスコープを含めた得意分野の階層を示します．

また，価格の違いについては，おおよそ**図2**のようになっています．この表は，導入時のおおよその価格帯について記載しています．その後のバージョンアップやサポート費用については各社まちまちの対応となっており，トータルコストではないことに注意してください．

〔図1〕解析分野

〔図2〕アナライザ別価格帯

	価格帯(円) 100万 200万 300万	備考
バスモニタ（～FS）		フルスピードまでの対応
アナライザ（～FS）		フルスピードまでの対応
バスモニタ（～HS）		ハイスピードまで対応 市場にはほとんどなし
アナライザ（～HS）		ハイスピードまで対応
アナライザ＋ジェネレータ		ハイスピードまで対応

2 USBアナライザを使用する場面

USBアナライザを使用したUSB機器開発が一般的になったといっても，USBアナライザが万能なわけではありません．とくに，電気的な特性(USB規格Chapter7)も確認したい場合などは，やはり餅は餅屋で，オシロスコープなどを活用すべきでしょう．たいせつなのは，USBアナライザがそれら他の測定器と連携する機能があるかどうかという点です．連携機能を使ってそれぞれの測定器や開発ツールの得意分野でデバッグを行い，開発効率を上げるという方法です．

また，実際に使用する場合に注意すべきなのが，USB機器を接続するケーブルの長さです．富士通デバイスでは，IEEE1394では世界で初めてトポロジー内でノードにならないNon-Node機能をもったアナライザを開発しました．しかし，USBの世界では，アナライザがUSBのバストポロジに存在しないのがあたりまえです(バス上に存在するとUSBホストからのデータ転送が異なってしまうため)．

したがって，実際のUSB機器使用時にはPoint to Pointで接続される場所にアナライザを挿入するため，電気的にまったく影響を与えないとはいいきれません(実際，Compliance Testで使用される治具でさえも，電気的な影響が少なからずあるともいわれている)．それゆえ，アナライザを接続する際に，不必要に長いケーブルを使用すべきではありません．

〔写真1〕USB ZERONE外観

3 USB ZERONE概要

ここで，富士通デバイス社製USBアナライザ"USB ZERONE"(**写真1**)について概要を紹介します．このUSB ZERONEは，IEEE1394アナライザとして実績のあるZERONEと同じ，

〔表1〕USB ZERONEの仕様

特　徴	トレースデータの高速表示 操作性に優れたGUI 小型・軽量
キャプチャメモリ	256Mバイト
主要機能	●トレース可能パケット速度：ハイスピード(480Mbps)，フルスピード(12Mbps)，ロースピード(1.5Mbps) ●トリガ機能：3ポイント，3レベル，3シーケンスの組み合わせで設定 ●トリガ要因：各種PID(PIDに応じた詳細設定が可能)，データ長，外部トリガ ●リアルタイムフィルタ：アドレス＋エンドポイントまたはデバイスクラスでの指定が可能 ●ファイル出力機能：HTMLファイル出力，指定エンドポイントデータ抽出(テキスト，バイナリ) ●解析機能：フレームごとのデータパケット遷移をグラフ表示 ●カスタマイズ機能：デコード文字列，表示カラム，表示パケット ●ファイル保存機能：トレースデータ，トレース条件，表示環境設定
コネクタ	●ターゲット接続：USBシリーズA × 1，USBシリーズB × 1 ●制御PC接続：USBシリーズB × 1 ●その他：拡張コネクタ(20ピン)，電源用コネクタ
表示装置	LCD(16文字 × 2行)，LED(電源：緑，トレース：橙)
外形寸法・重量	148mm(W) × 210mm(D) × 44mm(H)，650g
制御PC	対応OS：Windows98/98SE/2000/XP

ZERONEファミリの一つです．**表1**にUSB ZERONEの仕様概要を示します．

USB ZERONEは自身ではUIをもたないため，制御用のPCと接続して使用します．PC側の制御ソフトウェアの画面を**図3**に示します．USB ZERONEの制御ソフトウェアは，エクスプローラウィンドウ，トレースウィンドウ，イベントウィンドウの三つのウィンドウから構成されています．エクスプローラウィンドウは，二つのタブがあり，保存したデータファイルを開いた際にファイルの情報などを表示するプロジェクトタブとトリガ設定情報などを表示するアナライザタブがあります．

トレースウィンドウには，キャプチャしたデータをリスト形式で表示し，イベントウィンドウにはキャプチャデータをグラフィカルに表示します．表示されたパケットをダブルクリックすることで，パケット詳細ダイアログが開き，そのパケットの詳細を見ることができます．

4 USBアナライザ使用方法

USBアナライザは簡単にいうと，“キャプチャ”し，キャプチャしたデータを“トレース”するものです．冒頭でキャプチャとトレースという用語をわざわざ区別したものそのためです．その他の機能は，それぞれキャプチャもしくはトレース機能を使いやすくするために補助するといった，付加価値的な機能といっても過言ではないでしょう．

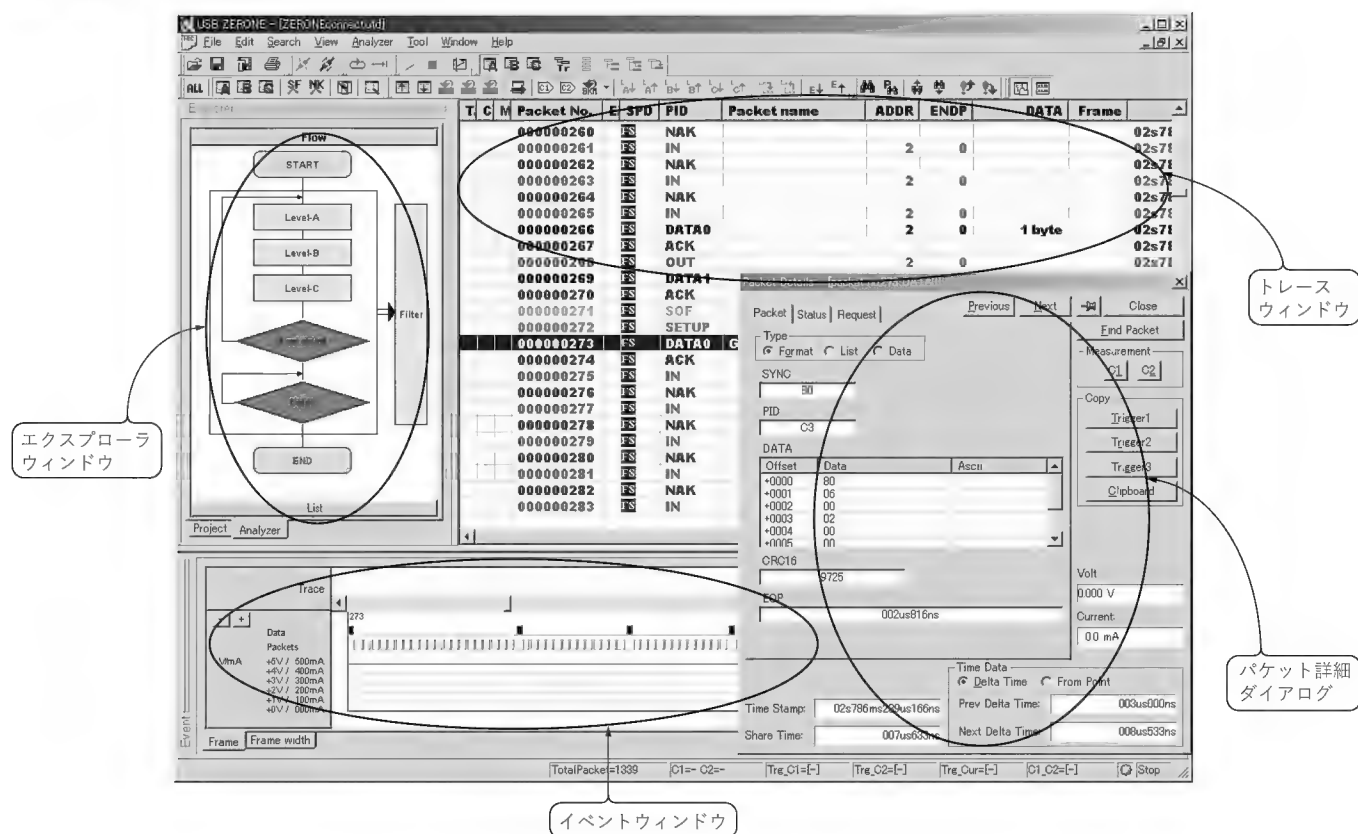
キャプチャ機能に付随する機能がトリガ機能，リアルタイムフィルタ(キャプチャフィルタ)機能です．また，トレースする際に付加価値を与えるおもな機能として，検索機能やデータ転送状態をグラフィカルに表示する機能をはじめとし，上位層レベルやデータ内容の翻訳(デコード)機能，表示フィルタ機能などが挙げられます．

その他の機能は，効率よくトレースするためのユーティリティ的な機能(時間・帯域測定機能など)であったり，制御ソフトウェアとしての補助的な機能(カスタマイズ機能など)です．

● キャプチャする

キャプチャする際にトレースしたい部分が特定できていない場合，もしくは，とりあえずデータの流れを見たい場合などがあると思います．そういった場合，USB ZERONEでは，“Quick Trace”という機能を使用します．この機能は面倒な設定をすることなく，USBバス上に流れるデータをキャプチャする機能です．この機能はアナライザメーカー各社がまちまちの名前を付けていますが，この機能はたいていのアナライザに装備されています．

USB ZERONEではさらに，Quick Traceにユーザーが停止するまでキャプチャし続ける“Free Run”モードと，設定したトレースメモリ容量分キャプチャしたら自動的に停止する“Buffer

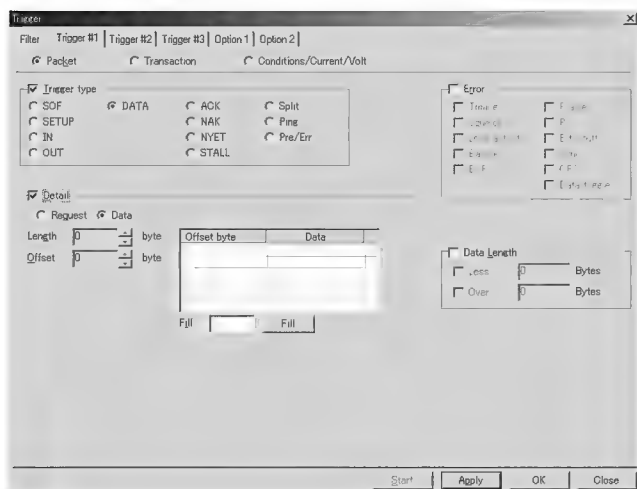
〔図3〕制御ソフトウェア画面

Full Break"モードを用意しています．また，ツールバーにボタンを用意しているため，1クリックでキャプチャすることが可能となっています．

逆に，トレースしたい事象が特定できている場合は，トリガを設定してキャプチャします．トリガには，PIDやデータの中身(データペイロード)などを指定します．USB ZERONEでは，トリガとして，PID，トランザクションの状態などが設定でき，選択した項目に応じた詳細な設定ができます．**図4**にトリガ設定画面を示します．

さらに，複雑な事象をトリガとしたい場合，設定したトリガを複数組み合わせて使用します．**図5**にUSB ZERONEのシーケンス設定画面を示します．USB ZERONEでは，トリガを3点まで設定でき，それらのトリガを使用して3レベルまでシーケンスを設定できます．それぞれのレベルでIF，ELSE IF，ELSEの3階層の条件分岐が指定できます．

これらトリガ設定やシーケンス設定は，ユーザーが容易に設定できるよう，グラフィカル部品を使用するなどアナライザメーカー各社が工夫を凝らしているところです．しかし，かゆいところに手が届くような詳細な設定も可能でなければ意味がありません．また，開発が進むにつれて，解析した事象が複雑になっていくと思われます．したがって，このトリガ機能を使いこなすことが，効率の良いデバッグを行うための近道でしょう．

● トレースする

次にキャプチャしたデータをトレースしていくわけですが，デバッグ場面や解析したい事象によって使用する機能はさまざまです．たいていの場合に使用する機能が，検索機能やマーク機能，ジャンプ機能でしょう．確認したいパケットやUSB機器のデータを検索したり，トリガ地点や設定したマーク地点にジャンプしてデータを確認，比較などを行ってデバッグを進めていきます．その他，デバッグ場面に応じて数々の解析機能や翻訳(デコード)機能，ユーティリティ的な機能を使用します．これらの機能の代表的なものを次に詳しく解説します．

5 USBアナライザの活用場面

USB機器開発において，アナライザを活用する場面は大きく二つあります．一つはUSB機器をPCなどのUSBホストに接続した際に行われるエニュメレーション(Enumeration)シーケンスをデバッグする場合です．このエニュメレーションが正しく行われないと，USB機器は目的の動作(データ転送)を行うことができません．したがって，もっとも重要なデバッグポイントであるといえます．もう一つのアナライザ活用場面は，パフォーマンスの向上を図る場面です．

それぞれの場面で使用するアナライザの機能と使用方法を解説します．

● エニュメレーションシーケンスのデバッグ

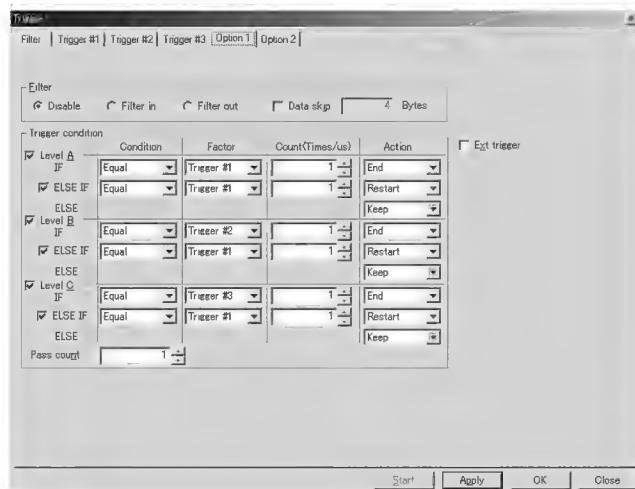
エニュメレーションシーケンスのデバッグで使用する機能は，なんといってもアナライザの翻訳機能です．標準リクエスト(**表2**)をデコードしたり，返送されたディスクリプタ(**表3**)をデコード表示する機能です．この翻訳機能を使って，リクエストに対して正しく返答しているかを確認していきます．

また，上位レベルでの表示機能も役に立ちます．これは，通常パケット単位で記録されるデータをToken-Data-Handshakeというトランザクション単位で表示する機能です．さらにトランザクション単位のデータを，たとえばコントロール転送で使用されるSETUP Stage-DATA Stage-STATUS Stageというトランスファ単位で表示することも可能です．

図6にUSB ZERONEでトランザクション(トランスファ)表示した例を示します(USB ZERONEはトランザクション＋トランスファを同時に表示する)．参考までに**図7**にWindowsのエニュメレーションが完了するまでのフローの一例を示します．

なお，翻訳機能については，一般的なアナライザであれば，標準リクエストと標準ディスクリプタには対応しています，各社おもなデバイスクラスへ対応したり，ベンダリクエストのデコー

〔図4〕トリガ設定画面

〔図5〕シーケンス設定画面

〔表2〕標準リクエスト

bRequest	Value
GET_STATUS	0
CLEAR_FEATURE	1
予約	2
SET_FEATURE	3
予約	4
SET_ADDRESS	5
GET_DESCRIPTOR	6
SET_DESCRIPTOR	7
GET_CONFIGURATION	8
SET_CONFIGURATION	9
GET_INTERFACE	10
SET_INTERFACE	11
SYNCH_FRAME	12

〔表3〕標準ディスクリプタ

Descriptor Types	Value
DEVICE	1
CONFIGURATION	2
STRING	3
INTERFACE	4
ENDPOINT	5
DEVICE_QUALIFIER	6
OTHER_SPEED_CONFIGURATION	7
INTERFACE_POWER	8

ドを設定可能にするなどの機能拡張を行っています.

● パフォーマンス向上のための解析

USBアナライザには，USB機器のデータ転送に関してパフォーマンスを向上させる手助けをするという重要な機能もあります．これは，アナライザメーカー各社が工夫を凝らしている点であり，各メーカーの特色が出ている点です．ここではUSB ZERONEに限った機能を述べます.

USB ZERONEでは，パフォーマンス向上のための解析に最適な“Frame Analyzer”機能が備わっています．この機能は，指定したUSB機器のデータパケットを集計し，フレームごとにグラフィカルに表示(**図8**)するものです．グラフの縦軸はフレーム，横軸はSOFパケットからのオフセット時間を示しています．また，データ転送に成功した(ACKが返送された)データパケットは黒，失敗した(NAKが返送された)データパケットは赤く表示します．また，グラフ内の空白部分はUSBホストからデータ転送(または転送要求)がなかったことを示します．このグラフで，赤く表示された部分や空白の部分の原因を追求することで，パフォーマンスの向上が図れるというわけです．このFrame Analyzer機能は，Isochronous機器用に集計したデータをヒストグラム形式での表示にも切り替えることができ，フレーム内のデータパケットが送出されたオフセット位置の統計も確認できます.

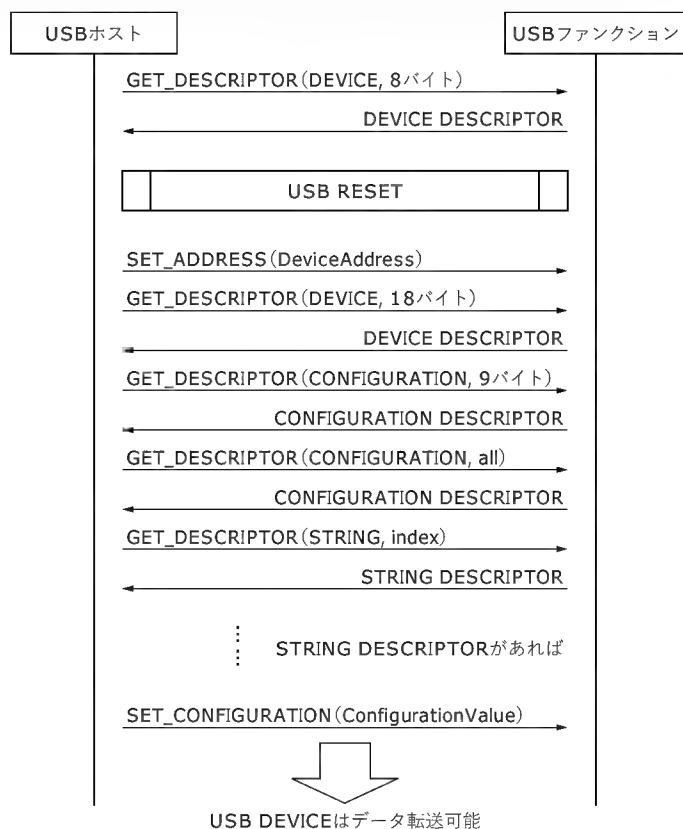
● その他のデバッグ

その他にアナライザを活用する重要な場面としては，データ

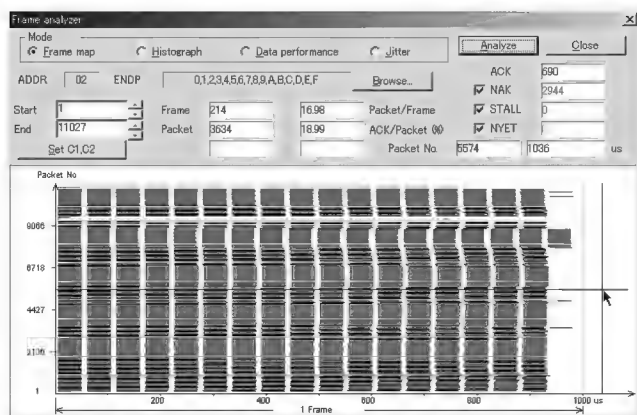
〔図6〕トランザクション表示

ZERONEconnect.utd

Transaction	Mark	Pkt[Trns] No.	SPD	PID	Packet name	ADDR	ENDP
GET_DESCRIPTOR [Device]		[000000001]			GET_DESCRIPTOR [Device]	0	0
SETUP [DATA0] [ACK]		[000000002]			GET_DESCRIPTOR [Device]	0	0
SETUP		000000016	FS	SETUP		0	0
DATA0		000000017	FS	DATA0	GET_DESCRIPTOR [Device]	0	0
ACK		000000018	FS	ACK			
IN [DATA1] [ACK]		[000000018]				0	0
IN		000000050	FS	IN		0	0
DATA1		000000051	FS	DATA1		0	0
ACK		000000052	FS	ACK			
OUT [DATA1] [ACK]		[000000019]				0	0
OUT		000000053	FS	OUT		0	0
DATA1		000000054	FS	DATA1		0	0
ACK		000000055	FS	ACK			
SET_ADDRESS		[000000020]			SET_ADDRESS	0	0
SETUP [DATA0] [ACK]		[000000021]			SET_ADDRESS	0	0
IN [DATA1] [ACK]		[000000029]				0	0
GET_DESCRIPTOR [Device]		[000000030]			GET_DESCRIPTOR [Device]	2	0
SETUP [DATA0] [ACK]		[000000031]			GET_DESCRIPTOR [Device]	2	0
IN [DATA1] [ACK]		[000000054]				2	0
IN [DATA0] [ACK]		[000000063]				2	0
IN [DATA1] [ACK]		[000000073]				2	0
OUT [DATA1] [ACK]		[000000074]				2	0
GET_DESCRIPTOR [Configuration]		[000000075]			GET_DESCRIPTOR [Configuration]	2	0
SETUP [DATA0] [ACK]		[000000076]			GET_DESCRIPTOR [Configuration]	2	0
IN [DATA1] [ACK]		[000000096]				2	0
IN [DATA0] [ACK]		[000000108]				2	0
OUT [DATA1] [ACK]		[000000109]				2	0
GET_DESCRIPTOR [Configuration]		[000000110]			GET_DESCRIPTOR [Configuration]	2	0
SETUP [DATA0] [ACK]		[000000111]			GET_DESCRIPTOR [Configuration]	2	0
IN [DATA1] [ACK]		[000000131]				2	0

〔図7〕エニュメレーションフロー

※上記はデバイスドライバインストール済みの場合

転送の中身自体が正しいか確認するという点でしょう．この点については，転送元のデータとキャプチャデータを比較するという作業が必要になります．USB ZERONEでは，“Export Data”機能に“Endpoint Data”ファイルを作成する機能があります．これは，指定したUSB機器の指定したエンドポイントのデータペイロードのみを抽出してファイル化する機能です．ファ

〔図8〕フレームアナライザ

イル形式は，バイナリ形式とデータダンプ表示のテキスト形式が選択できます．この機能を使うことによって，容易に転送元のデータと比較することが可能になります．

また，自身で解決できない問題点に遭遇した場合，キャプチャデータだけをやり取りし，他の開発者などに相談・協力を仰ぐ場合があります．アナライザメーカーではたいていの場合，制御ソフト(Viewer)のデモ版やサンプル版を無償配布しており，アナライザ本体が接続されていなくてもキャプチャしたデータを見ることができます．USB ZERONEでは，キャプチャデータをHTMLファイルとして保存する機能があります．

6 USBアナライザの選択

アナライザの選択にあたって重要な点は，まず第1にそのアナライザが得意とする分野が自分の使用目的と合っているかという点，第2に価格となるのではないでしょうか．使用目的に合っていなければ，使い物になりません．また，キャプチャ機能のみの安価なバスモニタであっても，キャプチャデータを加工するスキルと時間があれば，高度な解析が可能です．それらの時間を短縮したいなどの要求があるのなら，多少高価になっても高度な解析のできるプロトコルアナライザを選択したほうが良いでしょう．

そして次に，サポートやカスタマイズの可否および対応なども重要視すべき点だと思われます．日本語マニュアルやヘルプの必要性，制御ソフトの言語やプラットホームへの対応などが検討項目となります．

パケットジェネレータの必要性についても検討項目として挙げられるかもしれません．USB機器を開発するのであれば，USBホスト側のジェネレータが欲しいところでしょう．しかし，開発が進むとUSBホストとしては実使用環境(PC)でデバッグすることになるため，USBホスト側のジェネレータは使用期間が限られているといえます．そのような場合は，USB IF(Implementers Forum)のWebサイトに"SSTD(Single Step Transaction Debugger)"などが公開されており，そのようなツールを有効に使うのも一つの手です．

しかし，LSI開発などの場合は，そうもいかない場合もあるでしょう．LSI開発の際，ジェネレータに求められるのは，データを送出する時間関係を自由に設定できたり，電圧値を規格の上限・下限値に設定できる機能ではないでしょうか．そのような設定が可能なジェネレータはほとんどないのが現状です．オシロスコープやロジックアナライザなどはUSBの測定に特化したパッケージを用意してあります．それと同様にシグナルジェネレータにUSB信号送出用のパッケージを用意されるのが待たれるところです．

おわりに

アナライザメーカーに対しては，機能的なものをはじめとして，どんどん要望を出すべきでしょう．ニーズがなければアナライザも機能は向上しませんし，アナライザメーカーにとっては，役立つアナライザ開発の大きなヒントになります．

最後に，USB関連の情報が充実しているWebサイトを紹介します．掲示板などには，特定のデバイスや機器の情報が多数掲載されているものもあります．これらの他に，各LSIメーカーのWebサイトもUSB規格解説などの情報が充実しています．このような情報も活用してUSB機器開発に役立ててください．

本稿がアナライザを十二分に活用し，開発期間の短縮に少しでも役立てば幸いです．

■参考URL

- USB Implementers Forum：USB規格に関する公式Webサイト
  http://www.usb.org/
- Intel：Intel社のUSB技術情報サイト
  http://developer.intel.com/technology/usb/
- USB Design by Example：INTEL PRESS発行 USB Design by Example(John Hyde氏著)の連携Webサイト
  http://www.usb-by-example.com/
- Microsoft(USB Technology)：Microsoft社のUSB技術情報サイト
  http://www.microsoft.com/hwdev/bus/USB/default.asp
- エクスカル：国内でCompliance TestをしているUSB-IF認定テストラボ
  http://www.xxcal.co.jp/index.html
- USB Man：各種USB機器の製品情報サイト
  http://www.usbman.com/
- USB Stuff：各種USB機器の製品情報サイト
  http://www.usbstuff.com/
- Linux USB：Linux上でUSB機器をサポートするための技術情報サイト
  http://www.linux-usb.org/

■USB ZERONE問い合わせ先

富士通デバイス(株)　システム製品販売部
TEL：03-5434-0386　FAX：03-3490-9803
E-mail：usb@fdi.fujitsu.com

たにもと・かずとし　富士通デバイス(株)

組み込みプログラミングノウハウ入門

第10回 時相論理とプログラム検証のはなし(その2)

藤倉俊幸

はじめに

前回から，Dekkerのアルゴリズムのようなマルチタスクでかつ無限に走り続けて終了しない，普通のテストができないプログラムをどのように検証するかを説明している．前回は時相論理式の解釈についてあまり詳しく説明しなかったので，今回は，時相論理式の解釈や背景になるモデルについてもう少し詳しく説明する．

1 論理式

論理式(logical expression)といえば，プログラミングをする人にとっては，Boolean型変数と論理演算子&&とか||を使って書いた式のことで，評価結果がtrueかfalseになる式のことである．Boolean型というのは昔のC言語にはなかったので自分で定義して使っていたが，C++や新しいC言語[注1]では標準型として定義されている．

ただし，名前がboolだったり_Boolだったりする．その実態は整数で，値としてのfalseは0で，trueは0以外が一般的である．これだけではtrueは"真実"，falseは"正しくない"，という世間一般の感覚とは何の関係も見出せない．つまり，単なる値の割り当てである．たとえば，成功すると0を返すシステムコールなどがあると，falseということと成功ということをどう解釈し理解すればよいのかとまどうことになる．&&や||の論理演算子のほかに，比較演算子(>や<)と大小関係をもった整数型変数や定数などで論理式を作って，

0 > 2

を評価すると0となり，

2 > 0

を評価すると1になるので，世間一般の感覚と一致することができる．論理式を解釈するためには，プログラムの世界だけではダメで，この場合のように外部の構造，この場合には整数の体系が必要になる．そして，その体系の意味と式の表現が一致することが必要になる．

注1：ISO/IEC 9899：1999 - Programming Language C，略称C99.

この「一致する」ということが重要で，私たちが正しいと思っていることが論理式で表現してtrueになることをその論理式の体系の完全性(completeness)と呼び，逆にtrueになる論理式は私たちが正しいと思っていることと一致することを健全性(soundness)と呼ぶ(**図1**)．健全で完全な体系であることが重要である．完全でない論理体系は正しいことを表現できないので半端でいい加減な感じで，健全でない論理体系は正しくないことまで証明してしまうので危なっかしくて使えない．論語の「子の日わく，学んで思わざれば則ち罔(くら)し．思うて学ばざれば則ち殆(あや)うし」のようなものである．

何か正しいと思っていること，あるいは思いたいことを論理式で表現して，その論理式をたとえばド・モルガンの定理などで変形しても，表現の形が変わっただけで意味は変わらないことを保証するのが健全性と完全性である．ディジタル回路を論理圧縮できるのも表現の仕方や圧縮手順が電気回路という物理的構造と物理法則に対して健全で完全だからである．

プログラミングよりもう少し数学的な分野とか人工知能関係では論理式(formula)というと，いろいろ分類があるが，いちばん簡単なのは，命題記号(propositional letter)，あるいは単に命題を以下の論理結合子(logical connective)でつなげた命題論理式(prepositional formula)である．論理結合子は論理演算子と呼ばれることもある．別の記号が使われることもあるので，別の記号を後に並べておく．

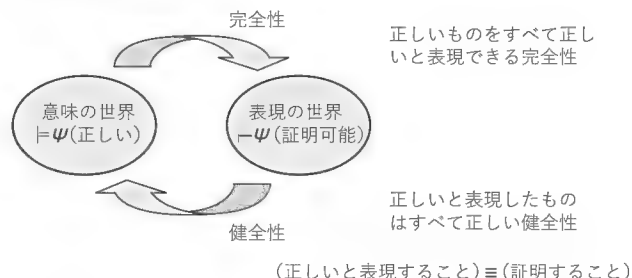
¬　(否定，negation)〜　!

∧　(連言，conjunction)　&

〔図1〕健全性と完全性

正しいものをすべて正しいと表現できる完全性

正しいと表現したものはすべて正しい健全性

(正しいと表現すること)≡(証明すること)

※論理記号の意味：⊨(正しい)，⊢(証明可能)，Ψは論理式

∨ （選言，disjunction）｜

→ （含意，implication）⇒ ⊃

↔ （同値，equivalence）⇔ ≡

論理式を読むときは，たとえば∧は“アンド”とか“および”，¬は“ノット”などと普通の呼び方をする．そして文章中に書くときに連言とか否定などを使う．そうしないと接続詞なのか何なのかわけがわからなくなる．時相命題論理式（prepositional temporal formula）では，さらに次の時相演算子（temporal operator）を使用する．時相論理式を読む場合には，□は“ずっと”とか“ボックス”，◇は“いつか”とか“ダイヤモンド”などと呼ぶことが多い．

□ 必然（necessity），ずっと（always）

◇ 可能（possibility），いつか（eventually）

xとyをtrueかfalseの値をとる命題として，xとyを上記の演算子でつなげた論理式もxとyの値に対応してtrueかfalseの値をとる．演算子ごとにどのような値を取るかを**図2**にまとめた．⊕，↑，↓などのロジック回路でよく使う演算子も加えてある．

単純な論理式は演算表（**図2**）を使えば計算できるが，次のような複雑な論理式の値は，どのように計算すればよいのだろうか．

$$p \to q \leftrightarrow \neg q \to \neg p$$

演算表だけでは複雑な式の評価（evaluate）はできないので，演算子の優先順位（precedency）を決めなければならない．優先順位は否定などの単項演算子の優先度が高く，後は**図2**の左側が高く右側が低い．そして，優先順位の高いほうから計算する．同一優先度の場合は，右から計算する．たとえば，$p \wedge q \wedge r$は$p \wedge (q \wedge r)$となる．このような計算手順も表現の世界（**図1**参照）に属する取り決めで，文法と呼ばれる．健全性完全性をいう場合には，この文法も含めたうえで考えなければならない．上の式の場合は，

$$((p \to q) \leftrightarrow ((\neg q) \to (\neg p)))$$

となり，内側の括弧から計算していく．計算の順を図で表現すると，**図3**のようになる．計算順の図を上下逆さまにすると，構文解析などで出てくる構文木（formation tree）になる．ここで計算とは，構文が明らかになった後で，命題pとqにtrueかfalseを割り当て（assignment）て，式全体の真偽を解釈することである．実際にpとqのすべての組み合わせに対して計算すると，**図3**の表のようになる．

この式は，「pならばqである」とその対偶「qでなければpでない」が同値であるという意味なので，常に真になっている．このような論理式は恒真（tautology，valid）といい，≡で表す場合もある．つまり常に成立する⟷を≡で表す．逆に，常に偽になる論理式は充足不能（unsatisfiable）と呼ばれる．その他の一般の論理式は，命題への真偽の割り当て方によって真になったり偽になったりする．たとえば，先ほどの式を少し変えた，

$$p \to q \leftrightarrow \neg p \to \neg q \quad \cdots\cdots (1)$$

の計算は，**図4**のような結果になる．

つまり，式(1)がtrueになるのは，p=T，q=Tとp=F，q=Fの二つの場合である．命題への真偽の割り当て方によってtrueになる論理式を充足可能（satisfiable）と呼ぶ（**図5**）．

充足可能な論理式を真にする割り当てを，その論理式のモデルと呼ぶ．ソフトウェア開発ではモデルという言葉がよく出てくるが，論理の世界では論理式を真にする割り当て全体の集合をモデルと呼ぶ．

ある論理式が与えられたときに，その論理式のモデルを探したり，恒真かどうか調べるもっとも単純な手順は，**図3**や**図4**のような真理表を作ることである．ただし，手順は簡単だが力仕事である．それで，人工知能分野ではいろいろな方法が開発されてきた．ヒルベルトシステム（hilbert system），ゲンツェンシステム（gentzen system），タブロー計算（tableau calculus），

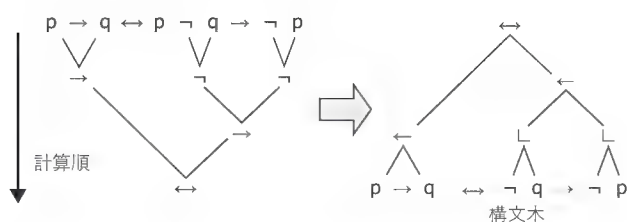
〔図2〕論理演算表

x	y	∧	↑	∨	↓	→	↔	⊕
T	T	T	F	T	F	T	T	F
T	F	F	T	T	F	F	F	T
F	T	F	T	T	F	T	F	T
F	F	F	T	F	T	T	T	F

T：true，F：false　　↑ nand　　↓ nor　　⊕ exclusive or

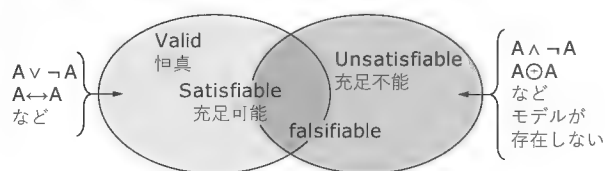
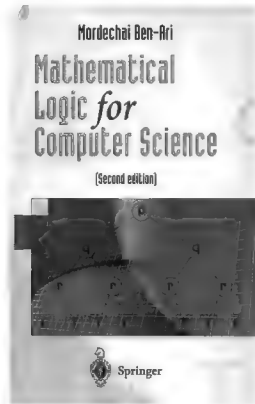
〔図3〕p→q⟷¬q→¬pの計算

p	q	p→q	¬q→¬p	p→q↔¬q→¬p
T	T	T	T	T
T	F	F	F	T
F	T	T	T	T
F	F	T	T	T

T：true，F：false

〔図4〕p→q⟷¬p→¬qの計算

p	q	p→q	¬p→¬q	p→q↔¬p→¬q
T	T	T	T	T
T	F	F	T	F
F	T	T	F	F
F	F	T	T	T

T：true，F：false　　モデル

〔図5〕論理式の分類

〔図6〕
ベン・アリの本

分解原理(resolution principle)などである．分解原理はPrologなどの論理プログラムの基礎となった．Windowsで使用できるPrologの処理系としては，

http://www.swi-prolog.org/

からダウンロードできるSWI-Prologが使いやすい．この処理系をインストールした後，前回紹介したBen-Ariの本[注2]（**図6**）のホームページ，

http://stwww.weizmann.ac.il/g-cs/benari/books.htm#ml2

から，本の中で紹介しているPrologプログラムをダウンロードする．力仕事はコンピュータにまかせるのがいちばんなので，ダウンロードしたプログラム(`mlcs-src.zip`)の中の`tt-t.pl`を起動する（**図7**）．`tt-t.pl`には，与えられた命題論理式の真理表を出力してくれる`create_tt()`が含まれる．たとえば**図4**の例であれば，

```
?- create_tt(p -> q <-> ~p -> ~q).
p->q<-> ~p-> ~q  p  q  value
                 t  t   t
                 t  f   f
                 f  t   f
                 f  f   t
Yes
```

といった具合に計算してくれる．`mlcs-src.zip`の中には，時相論理式に対してタブロー計算を行うプログラムなど，力仕事をやってくれるものが含まれている．ただし，この記事を読んだだけでは，これらのプログラムを使いこなすのは難しいかもしれない(Ben-Ariの本を買うしかない)．

2 時相論理

● 様相論理と時相論理

真実にもいろいろな相(mode)がある．必然的な真実とか，たまたまの真実とか，今日だけの真実とか，永遠の真実などである．このような相を□や◇で表す論理を，様相論理(Modal logic)と呼ぶ．時相論理は，時間を様相(modality)とするこのような論理の仲間である．その他の様相としては，必然性と可能性，全員が知っている知識と誰かが知っている知識などがある．たとえば，□Pの意味としては，

Pは必然的に真である．
Pは常に真であろう．
Pは真になるべきである．
エージェントはPを信じている．
エージェントはPを知っている．
プログラムの任意の実行の後Pが成立する．

がある．そして◇Pは，¬□¬Pの意味なので．□P：「Pは必然的に真である」を否定した¬□P：「Pは必然的に真でない」とは，「Pは偽になる可能性がある」であり，したがって¬□¬Pの意味は「Pの否定は偽になる可能性がある」だから，「Pになる可能性がある」ということになる．「エージェントはPを知っている」の場合の◇Pは，「エージェントはPでないことを知らない」，つまり「エージェントの知識の範囲ではPを否定できない」という意味になる．仕様打ち合わせのミーティングで，「ある機能を実装すべきである」は□Pに相当し，「ある機能を実装してもよい」は◇Pに対応する．実装のレベルに関する相が一致しないと，後々もめることになる．もめてしまったときには様相論理式を立てて推論し，誤解を解かなければならない[注3]．

様相の意味の次に，様相の文法というか演算子としての□と◇について見てみたい．□と◇は単項演算子なので，否定¬と同一の優先度をもっている．同一優先度の場合は右から結合す

〔図7〕tt-t.plを起動した画面

注2：M. Ben-Ari, *Mathematical Logic for ComputerScience*, Springer, 2001.
注3：藤倉，『リアルタイム/マルチタスクシステムの徹底研究』，インターフェース増刊 TECH I vol.15，pp.224-230.

るので，$\Diamond \neg \Box p$ は $\Diamond(\neg(\Box p))$ の順に計算される．

$$\Box(\Diamond q \land \neg r \to \Box p)$$

の計算順は，

$$\Box(((\Diamond q) \land (\neg r)) \to (\Box p))$$

のようになる．一方，

$$\Box \Diamond q \land \neg r \to \Box p$$

の場合は，

$$((\Box(\Diamond q)) \land (\neg r)) \to (\Box p)$$

となり，まったく別の論理式になってしまう(**図8**)．

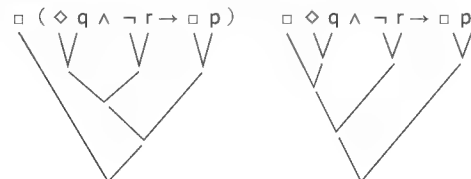
計算手順は，以上のように普通の論理式と同一である．では，意味はどうなるのだろうか．意味というのは，時相論理式の真理表とかモデルに対応するものである．「ずっと」とか「そのうち」などといっても，これらの物がハッキリと見えないと，何のことかわからない．普通の論理式の p や q は，命題や大小関係などだった．たとえば，$0 > -2$ とかであれば整数の世界が背景にあり，その背景を使って論理式の意味も明確になる．

このような背景となる数学的な構造をストラクチャと呼ぶ．ストラクチャが与えられると，論理式を解釈できるようになる．そして解釈した結果，論理式が真になるストラクチャ(の事例)があれば，そのストラクチャ(の事例)はその論理式のモデルになる．時相論理式の意味を考えるということは，時相論理式のストラクチャとモデルを考えることから始まる．で，結論を先にいってしまうと，時相論理式の場合，状態マシンと同様の表現になる．ということで，組み込みプログラムと相性がよい論理体系であることが予想される．

● クリプキ(Kripke)モデル

様相論理のストラクチャは，クリプキ(Kripke)モデルとかクリプキストラクチャと呼ばれている．クリプキモデルは，世界の集合 W と世界間の到達関係 R，各世界での(真になる)命題(p とか q)の割り当て L の三つのものから構成される．世界が状態マシンの個々の状態に対応し，到達関係が状態間の遷移関係に対応する．ただし，状態マシンとは違ってすべての状態が遷移関係で連結するとはかぎらない．各世界で命題の真偽が与えられるので，各世界で与えられた論理式が様相記号を含まなければ普通に評価をすることができる．様相記号を含んでいる場合は，遷移関係を見て評価することになる．たとえば，ある状態 w で命題 p が真であると L によって割り当てられていて，その状態から到達可能なすべての状態 w_i でも p が真であれば，w で $\Box p$ が真になる．すべてではないがいくつかの状態 w_j で真になれば，$\Diamond p$ が真になる．

〔**図8**〕 $\Box(\Diamond q \land \neg r \to \Box p)$ と $\Box \Diamond q \land \neg r \to \Box p$

$\Box(\Diamond q \land \neg r \to \Box p)$　　$\Box \Diamond q \land \neg r \to \Box p$

たとえば，次のようなクリプキモデルを考えてみる．

$W = \{w_1, w_2, w_3, w_4, w_5, w_6\}$

$R = \{(w_1, w_2), (w_1, w_3), (w_2, w_2), (w_2, w_3), (w_3, w_2), (w_4, w_5), (w_5, w_4), (w_5, w_6)\}$

$L = \{w_1 : \{q\}, w_2 : \{p, q\}, w_3 : \{p\}, w_4 : \{q\}, w_5 : \{\phi\}, w_6 : \{p\}\}$

このストラクチャ S = <W, R, L> を図で表現すると，**図9** のようになる．

図9 が普通の状態マシンと違うのは，すべての状態が連結していないことである．ある論理式 A がストラクチャ S の世界 w で成立することを，

$$S, w \models A$$

で表す．w だけ書いて S は省略されることもある．S だけ書いて w のほうを省略すると，S のすべての状態で成立することを表す．両方省略すると，後述する別の意味になる．状態マシンの言葉では，状態マシン S の状態 w で条件 A が成立するということである．また，成立しない場合は $\not\models$ で表す．たとえば，**図9** では，

$w_1 \models \Diamond q$　w_1 から到達可能な世界で q が成立する世界がある．つまり w_2．

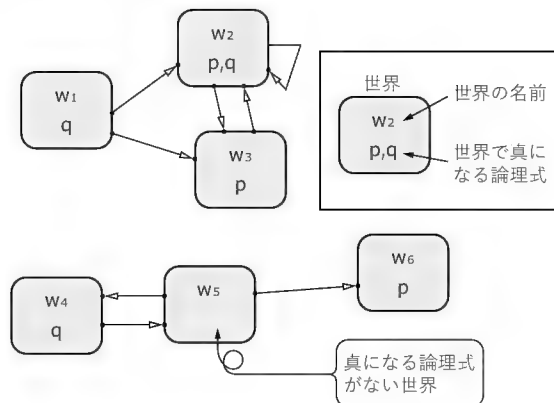
$w_1 \not\models \Box q$　w_1 から到達可能なすべての世界で q は成立しない．つまり w_3．

$w_5 \models \Box(p \lor q)$　w_5 から到達可能な w_4 で q が成り立ち，w_6 で p が成り立つ．

$w_5 \not\models \Box p \lor \Box q$　w_4 では p が成り立たず，w_6 では q が成り立たない．

$w_2, w_3, w_4, w_5, w_6 \models \Box p \to p$
　この時相論理式が成立する世界は，w_2，w_3，w_4，w_5，w_6 である．

最後の $\Box p \to p$ で，w_2，w_3，w_6 については p が常に成立しているので正しい．w_4，w_5 では $\Box p$ が成立しないのでやはり正しい．w_1 は，$\Box p$ は満足するが p を満たしていないので正しくない．w_1 から遷移可能な w_2 と w_3 で p が成立するので w_1 では $\Box p$ が成立すると考える．□ を「ずっと」と解釈すると，今も含めて

〔**図9**〕クリプキモデルの例

〔図10〕様相による恒真の違いの例

	様相の違い（□φの意味）					
	必然	将来常に	べきである	信じる	知っている	プログラム
□φ→φ	○	×	×	×	○	×
□φ→□□φ	○	○	×	○	○	×
◇φ→□◇φ	○	×	×	○	○	×
◇(true)	○	×	○	○	○	×
□φ→◇φ	○	×	○	○	○	×

↑ 時相論理（将来常に）

○は恒真であることを示す
φは論理式

〔図11〕□pと◇q

□：ある世界から到達可能なすべての世界で成立する
◇：ある世界から到達可能な少なくとも一つの世界で成立する
この到達可能とは何のことか？

〔図12〕様相による世界間の関係の違いの例

様相の種類（□φの意味）	R(x, y)の意味
必然	yはxの情報によって真になる
常に	yはxの将来
べきである	yはxの情報によって受け入れ可能
信じる	yはxにおけるエージェントの信念によって実現される
知っている	yはxにおけるエージェントの知識によって実現される
すべての実行後成立する	yはxにおけるプログラムの実行の結果になり得る

時相論理 → 常に

成立すると解釈するのが日本語として自然かもしれないが，現在pが成立していなくとも□pは成立する場合があることに注意する必要がある．これは後述するが，時間の概念の論理体系への入れ方に依存する．

w_6のようなどこにも遷移できない世界では，◇を使った論理式は成立しない．◇が成立するためには，次の世界の存在が必要である．Trueはすべての世界で成立する論理定数であるが，◇(true)のような定数式もw_6では成立しない．逆にいえば，◇(true)が真ということとその世界から遷移可能ということは，論理的に同一である．□と◇には双対関係があるので，逆にw_6では□(false)のような定数式でも真になる．

どのようなモデルのどのような世界でも真になる時相論理式Aを恒真と呼び，⊨Aと書く．たとえば，

$$\models \Box(\phi \to \psi) \to (\Box \phi \to \Box \psi)$$

がある（ϕ, ψは論理式を示す）．この式はスキームKと呼ばれる．□p→pは恒真ではないことを示したが，□pが「必然的にpは真である」という意味であれば，必然的に「pは真」でなければ話が通らない．つまり，健全で完全な議論ができない．しかし，□pを「エージェントはpを信じている」という意味で使用すれば，「pは真」とはいい切れない．つまり，何が恒真になるかは使用する様相によって異なってくる．

時間を様相とする場合，将来が現在を含むかどうかが問題になってくる．時相論理では含める場合と，含めない立場を取る場合がある．モデル検証で使用されるCTLという論理体系では含める立場を取る場合が多い．つまり，様相を決めても時間の扱い方によって，公理としての恒真をどのように決めるか，いろいろな立場がある．公理とは，ユークリッド幾何学で平行線は交わらないなどと証明なしで正しいと認めるもののことである．たとえば◇(true)の扱いは，時間に終わりがあるか否か，どちらの立場を取るかによって恒真となるかならないかが分かれる．◇(true)を常に真であると認めることは，常に次の状態が存在することを認めることであり，時計は永遠に止まらないというか，永遠が存在するというか，時間に終わりがないと認めることになる．このあたりは，自分が作ろうとするモデルに合うように決めればよい．**図10**に例を示したが，これは文献[注4]に紹介されている例である．たとえば時相論理では，□p→pを公理に加える場合のほうが多いように思う．□pと◇qについて，**図11**に示す．

次に，クリプキモデルのRについて調べてみよう．Rについては状態マシンの遷移のようなものと説明したが，数学的には代数における関係に対応する．代数における関係として考えると，反射的(reflexive)とか推移的(transitive)とかの性質を明確にしたくなる．これらの性質を考えることで，何を公理とすべきかがわかってくる．まず，各様相におけるRの意味の例，あるいは解釈の仕方を**図12**に示す．幸い，時相論理の場合は時間の流れと解釈できるので，Rの意味は比較的明確である．

では，時間の流れにおいて反射的とはどのようなことだろうか．これは，すでに出てきた□p→pが恒真となることである．ここでは説明していないが，CTLにおける時間は反射的な性質をもつことになる．□p→pの代わりにp→◇pで表すこともある．「現在は将来だ」といえる時間の流れは反射的である．別の言い方をすると，「将来は現在も含む」ということである．モデルとして考えた場合は，自己遷移R(x, x)は反射的である．時間の性質として反射的にするか，特定の状態の性質として反射的性質を導入するか2通りが可能である．

次に推移的とは，R(x, y)かつR(y, z)であればR(x, z)と

注4：M. Huth, M. Ryan, *Logic in Computer Science: Modelling and Reasoning about Systems*, Canbridge, 2002.

なることをいう．時間の流れとしては，「将来の将来は将来である」ということであり，時相論理では普通受け入れられる性質である．これは$\Box p \to \Box\Box p$あるいは$\Diamond\Diamond p \to \Diamond p$を恒真として受け入れることに対応する．$\Box p \to \Box\Box p$ということは，ある状態で$\Box p$をいうためには次の状態でpが成立するだけではダメで，その先の状態でもずっとpが成立しなければならない．そこで，推移的時間を使う場合，次の状態で成立することだけをいうためには$\circ p$を使う(**図13**)．$\circ$は，nextオペレータと呼ばれる時相演算子である．

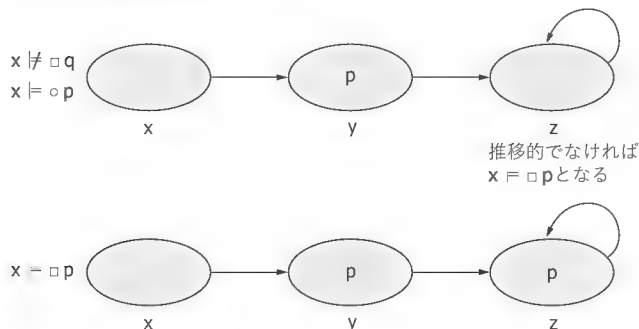
時間の扱いでもう一つ重要なのは，線形時間か分岐時間(Branching-time logic)かの違いである．線形時間の場合，クリプキモデルはただの一本道になる(**図14**)．クリプキモデルは<W, R, L>で指定されたが，<W, R>の部分をフレームと呼ぶ．このフレームが状態マシンとよく対応することを説明したが，線形時間では無限に続く一本の状態列であるフレームのみを使用する．つまり，普通の意味の状態マシンは必要なくて，時計があればよい．各世界(あるいは状態)は時刻によって指定できる．たびたび出てくるCTL(Computation tree logic)は，分岐時間を使用する体系である．分岐時間のほうが状態マシンを使用する並列プログラムの表現には向いているが，今回は線形時間を主に扱っている．

モデル検証では分岐時間であるCTLを使用する場合が多い．しかし，CTLに入るためにはもう少し準備が必要である．そこでとりあえず今回は，反射的で推移的な線形時間に基づくストラクチャを採用する．線形時間では，実質的に状態マシンに依存しないので状態数が多くなって利用できないということがない．

3 推論規則

要求仕様や設計仕様，実装仕様などを，論理式で表現すると何がよいのかという素朴な疑問がある．自然言語のあいまいさを排除できるということもあるがそれだけではなく，プログラムの正しさを証明できるというところが大きなメリットである．

● 公理系と推論規則

推論システム(deductive system)は，公理(axiom)と推論規則(inference rule)からなる．ある論理式がその推論システムで証明(proof)できることを，

$$\vdash A$$

と書く．

反射で推移的な線形時間に基づくストラクチャにおける推論システムとしては，次のものがある．

公理の集合としては，

1. $\vdash \Box(A \to B) \to (\Box A \to \Box B)$ 　$\Box$の分配
2. $\vdash \circ(A \to B) \to (\circ A \to \circ B)$ 　$\circ$の分配
3. $\vdash \Box A \to (A \land \circ A \land \circ\Box A)$ 　反射と推移
4. $\vdash \Box(A \to \circ A) \to (A \to \Box A)$ 　帰納
5. $\vdash \circ A \longleftrightarrow \neg\circ\neg A$ 　線形時間

に，命題論理における恒真の命題を時相論理式で置き換えたものを使う．それから，推論規則としては，三段論法(Modus Ponens)と一般化(Generalization)を使う．一般化は，$\vdash A$のときに$\vdash \Box A$を導く規則である．この意味は，$A \to \Box A$を公理に追加するということではなく，Aという論理式が証明できたとすれば，それは恒真だから，対象としているモデルのすべての状態でも成立するということである．

● 並列プログラムの検証

並列プログラムの検証をする推論システムの場合には，さらに並列プログラムを表現するストラクチャの詳細が必要になり，そのストラクチャにおける恒真を公理の集合に加える必要がある．

ストラクチャの詳細としては，クリプキモデルをベースとして状態と遷移をどう定義するかが問題となる．並列プログラムは，並列に動くプロセス$\{p_1, p_2, p_3, \ldots, p_n\}$から構成されるので，状態を定義するためには，これらのプロセスが実行中のステートメントやプログラムカウンタに相当するもの(ロケーション)を考慮する必要がある．それで状態は，プログラム内の各変数の値$(v_1, v_2, v_3, \ldots, v_m)$だけではなく，プロセスのロケーション$(l_1, l_2, l_3, \ldots, l_n)$とによって定義される．そして遷移は，一つのプロセスを適当に選んで1ステップ進めることと定義する．

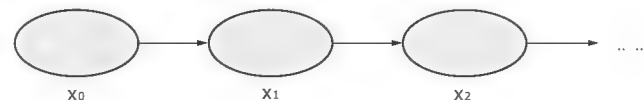
公理に新たに加えるのは，前回並行プログラムの実行仕様として紹介したものである．それを，**図15**に再掲する．

● ピータソンのアルゴリズム

前回はデッカーのアルゴリズムを例として使用したので今回はピータソン(Peterson)のアルゴリズムを使用する．**リスト1**にソースコードを示す．状態は(`C1`の値，`C2`の値，`Last`の値，`P1`のロケーション，`P2`のロケーション)で構成される．ロケーションはソースコード上に`NC1`，`Set1`，`Test1`，……などのコメントで示した．

この並列プログラムで，`P1`がクリティカルセクション`CS1`に

〔図13〕推移的時間モデル

〔図14〕線形時間フレーム

〔図15〕並列プログラムの実行仕様

プログラム	実行仕様
l_i：v:=式	$l_i \rightarrow \Diamond l_{i+1}$
l_i：if B then l_t：　S1 　else l_f：　S2	$(l \wedge \Box B) \rightarrow \Diamond l_t$ $(l_i \wedge \Box \neg B) \rightarrow \Diamond l_f$
l_i：while B do l_t：　S1; l_f：S2	$(l_i \wedge \Box B) \rightarrow \Diamond l_t$ $(l_i \wedge \Box \neg B) \rightarrow \Diamond l_f$

入れることを証明する．証明すべき論理式は，

$$Test1 \rightarrow \Diamond CS1$$

である．プログラム的にはTest1のWhile文が無限ループ($\Box$ Test1)にならないことを示せばよい．まず，

$$\Diamond C2 \rightarrow \Diamond (Last=2) \cdots\cdots (2)$$

を証明する．意味はvoid P2(void)のコードを見れば一目瞭然だが，C2がいつか真になるならばLastはいつか2になるということである．

1. $\vdash \Box NS2 \vee \Diamond Set2$
2. $\vdash \Box NS2 \rightarrow \Box \neg (Test2 \vee CS2)$
3. $\vdash \Box \neg C2 \vee \Diamond Set2$
4. $\vdash Set2 \rightarrow \Diamond (Test2 \wedge (Last=2))$
5. $\vdash \Box \neg C2 \vee \Diamond (Last=2)$
6. $\vdash \Box (\Box \neg C2 \vee \Diamond (Last=2))$
7. $\vdash \Box (\neg \Box \neg C2 \rightarrow \Diamond (Last=2))$
8. $\vdash \Box (\Diamond C2 \rightarrow \Diamond (Last=2))$

1は非クリティカルセクションに関するパターソンのアルゴリズムにおける定義で，二つのプロセスP1，P2は非クリティカルセクションに無限に留まることもできるし，非クリティカルセクションから出てクリティカルセクションに入ることを要求することもできるということを表している．2は非クリティカルセクションに留まっているのであれば，他のロケーションにはいないということである．3は，1と2と変数C2が真であることとプロセスP2がTest2またはCS2ロケーションにいることは同値である，

$$\vdash \Box (C2 \longleftrightarrow (Test2 \vee CS2))$$

から

$$\vdash \Box NS2 \rightarrow \Box \neg (Test2 \vee CS2)$$

$$\vdash \Box NS2 \rightarrow \Box \neg CS2$$

によって導かれる．4は，Set2からいずれはTest2に実行が進

〔リスト1〕ピータソンのアルゴリズム

```
typedef int bool;
#define True 1
#define False 0

bool C1 = False, C2 = False;
int Last = 1;


void P1(void)
{
    while(True) {
         /* ----- */                  /* NC1    */
         C1 = True;                   /* Set1   */
         Last = 1;
         while (C2 && Last == 1);     /* Test1  */
         /* ----- */                  /* CS1    */
         C1 = False;                  /* Reset1 */
    }
}

void P2(void)
{
    while(True) {
         /* ----- */                  /* NC2    */
         C2 = True;                   /* Set2   */
         Last = 2;
         while (C1 && Last == 2);     /* Test2  */
         /* ----- */                  /* CS2    */
         C2 = False;                  /* Reset2 */
    }
}
```

むという実行仕様である．5は3と4と，

$$\vdash \Diamond (p \wedge q) \rightarrow (\Diamond p \rightarrow \Diamond q)$$

から導かれる．6は5の一般化である．

無限ループにならないことの証明に戻る．

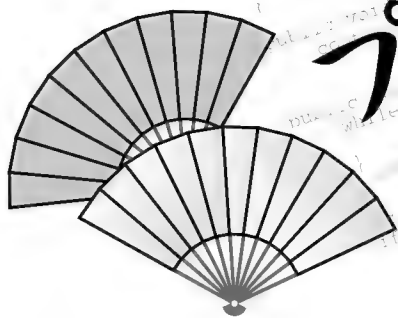
1. $\vdash \Box Test1 \rightarrow \Diamond (C2 \wedge (Last=1))$
2. $\vdash \Box Test1 \rightarrow \Diamond C2 \wedge \Diamond (Last=1)$
3. $\vdash \Box Test1 \rightarrow \Diamond (Last=2) \wedge \Diamond (Last=1)$
4. $\vdash \Box \Box Test1 \rightarrow \Box (\Diamond (Last=2) \wedge \Diamond (Last=1))$
5. $\vdash \Box Test1 \rightarrow \Box \Diamond ((Last=2) \wedge (Last=1))$
6. $\vdash \Box Test1 \rightarrow false$

1はwhile文でループする条件．3は式(2)を使った．

ループを抜けないと仮定すると，最終的に変数Lastが1でありかつ2であるという矛盾に至ったので，ループを抜けることが証明された．

おわりに

分岐時間を使用する場合の説明までできなかった．分岐時間についてはまた別の機会に説明したい．

ふじくら・としゆき　日本ラショナルソフトウェア(株)

宮坂電人

第2回 開放/閉鎖原則(前提知識編)

2000年問題と1000万円の見積もり

最近は世の中の変化が激しいせいか，ほんの少し前の出来事が，何だか遠い過去のように錯覚してしまうことがあります．プログラミングを商売にしている人は，「2000年問題」というのが少し前にあったことを記憶しているでしょう．あるいは2000年問題対応で悲惨な目にあったとか．でも最近はあまり聞かないので，すっかり過去の出来事と化しています．

メディアでも連日，2000年問題が取り上げられましたが，中にはずいぶん「怪しい」(!?)報道もありました．2000年問題を解決していない心臓ペースメーカーが止まるとか，ミサイルが誤発射するとか．結局，ミサイルは誤発射されず，ペースメーカーが止まって死ぬ人もおらず，大騒ぎしたわりには，ほぼ何事もなく21世紀を迎えられました．

ところで，そんな騒ぎに隠れて，笑える(といったら当事者に怒られそうだが)2000年問題もありました．ある中小企業で使用しているオフコン(office computer/オフィスコンピュータの略称)に2000年問題があり，そこまではよくありがちですが，その2000年問題を解決するためにオフコンメーカーから提案されたソフトウェアの修正見積もり金額が1000万円以上したという話です．バブル期ならともかく，デフレ不況な時期に中小企業が1000万円以上の大金を簡単に捻出できるわけがありません．

結局どうなったかというと，そこの社長はオフコンをすべて撤去し，代わりにWindowsパソコンで経理/事務システムを構築しました．しかもシステムのハード/ソフト込みで1000万円もしなかったそうです(おそらく200～300万円程度だろう)．そのTV番組を見ていた筆者は爆笑してしまいました．なぜオフコンメーカーは2000年問題の解決で1000万円以上の修正見積もりを出したのでしょうか．

2000年問題というのはよく知られているように，年号をプログラムで取り扱うとき，1995年，2001年の4桁ではなく95年，01年の2桁で行ったため，2000年以降，年号の逆転現象が起きて不具合が起きることです．なぜ2桁にしたかというと，昔のコンピュータはメディアやメモリが高価だったので節約するために2桁にしたとか，そもそも2000年以降まで同じソフトが使われることはないと油断したなど，いくつかの説があります．

だとしても，2桁だったのを4桁に増やすだけで1000万円以上かかる理屈がよくわかりませんでした．2桁決め打ちの箇所が多数あるのでプログラムの修正が手間どるだの，すでに2桁で記録しているメディアの変換で手間どるだの，といろいろ考えたのですが，だとしても，いくつかの手間はユーザーに「押し付け」ればよい話であり，たかだか2桁を4桁にプログラムを変更する程度で(といっては当事者は気分を悪くされるかもしれないが)，そんなに費用がかかるものなのかと思いました．筆者は組み込みのプログラムで極端にメモリやメディアをケチるプログラムをいろいろ作っていますが，2バイトで十分と思ったのが足りなくて4バイトに拡張する事態はしょっちゅう遭遇しますし，もちろんそのたびにユーザーに2バイトを4バイトに変更するので1000万円くださいなどといったことはありません(笑)．

冗談はさておき，思いもかけない事態でプログラムを修正することは日常茶飯事ですし，修正がやっかいで日程や費用が恐ろしく高くつく事態はプログラムの大規模化/複雑化によって，ますます増えているようです．TV番組の社長さんみたいに別のシステムに総入れ換えするドラスティックな手段で解決できればよいのですが，いつもそんなにうまくいく保証などありません．

『オブジェクト指向入門』について

今回は前回に引き続き，知っている人にとっては常識でも，そうでない人には聞いたことすらない「開放/閉鎖原則」(Open-Closed Principle)を取り上げようと思い，同原則が載せられている『オブジェクト指向入門』[注1]という，何とも直球的なタイトルの本を読み直しました．

この本はまだ世間がバブルで浮かれている時期に書かれていたものなのに，なかなか堅実で良いことが書かれているのです．

注1：バートランド・メイヤー著，アスキー，ISBN4-7561-0050-3．原題は*Object-Oriented Software Construction*．翻訳書は1990年発行で初版だが，原書はすでにSecond Editionが出ている．そちらはPrentice Hall，ISBN 0-13-629155-4．参考URLは`http://archive.eiffel.com/doc/oosc/`．

ただ当時は，まだまだオブジェクト指向の効能が理解されず，それどころかEiffelという，あまりなじみのないプログラミング言語の解説書と思われてしまったため，それほど読まれなかったようです．しかし，オブジェクト指向のメリットを期待してJavaやC++を導入したはいいが，まるで頓珍漢な運営で苦しんでいる今のプログラマにこそ読んでほしいと筆者は思います．なにしろ当時は，まだデザインパターンだのアジャイル開発だのが出る以前であり，オブジェクト指向とは何ぞや，その御利益は何なのかを一生懸命に説明しようとしている時期でした．そのせいか今読むと，わりとわかりやすいのです．

ソフトウェアの品質

『オブジェクト指向入門』は，翻訳されたもので700ページもの大著です．またEiffelの仕様解説や他のプログラミング言語との比較などの，純粋にオブジェクト指向そのものを説明していない箇所がいくつもあります．忙しくて全部を読んでいられない人にはPart1「問題と原則」だけでも十分にオブジェクト指向入門になるでしょう．今読んでも短いページ数で，ずいぶん本質をついている良い入門記事です．オブジェクト指向というと，やたら気張って難解な用語を並べ立てることと勘違いしている(?)記事もありますが，そのアンチテーゼともいえるでしょう．

第1章でソフトウェアの品質の定義が述べられています．これはオブジェクト指向以前の時代から考えられてきたもので，とくに目新しいものはありません．しかし技術者がつまづく，ある側面での分類が書かれています．それは，

- **外的品質要因：**ユーザーが，その品質を認められるような要因．処理速度，使い勝手，従来品との互換性など
- **内的品質要因：**専門家でないと，その品質を認められないような要因．ソースコードの読みやすさ，ソフトウェアモジュールの構成など

という二面での分類です．たしかにわれわれは，この二つをいっしょくたにすることで，ユーザーとの議論で齟齬(そご)を生じてきましたが，同じことは技術者同士でも生じてきました．ついつい内的品質要因のソースコードやモジュール構成に興味をもってしまいがちですが，処理速度が遅かったり，極端にメモリや資源を使いきるので動作しないもの，つまり外的品質要因で不具合のあるものはどうしようもありません．そのあたりを認識せずに，内的品質要因が良いのだからよいに決まっているという派と，外的品質要因にこだわる派で平行線をたどってしまったというわけです．

外的品質要因

『オブジェクト指向入門』では，外的品質要因として次の五つを挙げています．

- **正確さ：**ソフトウェア製品が要求/仕様に定義されたとおりに確実に仕事を行う能力
- **頑丈さ：**異常な状態でも機能するソフトウェアシステムの能力
- **拡張性：**ソフトウェア製品が仕様変更に容易に[注2]適応できる能力
- **再利用性：**ソフトウェア製品の全体または一部が，どの程度，新規ソフトウェア構築に再利用できるかを示す能力
- **互換性：**ソフトウェア製品相互の組み合わせやすさを示す能力

● 正確さ

いうまでもなく，これが最重要な品質要因です．問題は何をもって「正確」と定義するのかです．同書でも「システム要求を完全に形式的に表現することからして難しい」と書かれています．これは形式的な側面での正確さの定義の難しさですが，それ以前の問題としてソフトウェア製品を要求し，仕様を書いている当事者自身が正確ではありません．とくに，しょっちゅう仕様変更や追加を意味なく繰り返す人や組織の場合，人や組織そのものが不正確な特性をもっているのでやっかいです(笑)．

● 頑丈さ

正確の定義が難しいので，その反対の“異常な状態”の定義も難しくなります．ただし同書で論じているのはそんな哲学的議論ではなく，異常な状態でどうふるまうかです．異常事態のときは，へたな対応をせずに中断して，それ以上傷を深くしないように配慮できるかどうかの能力です．

● 拡張性

拡張性を向上させるために重要な二つの項目を同書で紹介しています．それは，

- **簡明さ：**複雑なアーキテクチャよりも簡単なアーキテクチャのほうが変更に適応しやすい
- **非集中性：**モジュールが独立しているほど変更の影響が狭い範囲にとどまりやすい

です．筆者の経験でいえば，簡明さはわりあいに意識されていて，複雑すぎて変更しにくい例は少なく，むしろ非集中性が守れていないものが多いように思います．たとえば，やたらにグローバル変数を使っていて，しかもその使い方がアクロバティックで，たった1箇所を変更しようとしても将棋崩しのように，そこから影響が連鎖反応を起こして他の場所へおよび，結局どうしようもなくて頭をかかえた経験をおもちのプログラマの方も多いと思います．

● 再利用性

つねに新規のソフトウェアを作成しているように思えても，経験を積めば積むほど似通った場面に遭遇することはありがちです．たとえば，ある情報を大きいもの順に表示するプログラムを作っているときに，そういえば以前にも大きいもの順に情報

注2：「容易に」が重要．徹夜や人海戦術によって仕様変更が達成できても，拡張性があるとはいいがたい．

を並べる処理をやったという“デジャヴ”に襲われることがあります．

標準ライブラリや特定問題向けフレームワークと呼ばれる再利用を考慮したソフトウェア製品は，そのような何度も遭遇するテーマをなるべく容易に解決するために存在します．たとえば，大きいもの順に並べる処理のためのソート処理の関数やクラスなどを提供してくれます．しかし，そういった製品はあくまで“平均的な標準”であり，すべてのプログラムをかなえてくれるものではありません．となると，個々の何度も遭遇するパターンに対処できるライブラリをおのおのの現場で準備しないと実現できないのですが，悲しいかな，納期に追われていたり，どうやって再利用性を実現できるのか具体的な方法がわからないという現状が多々あります．

● 互換性

ここでいう互換性とは，過去に作成した同じソフトウェアとの互換性ではありません．他のソフトウェアとの連携での互換性です．具体的には，他のソフトウェアが作成したファイルを取り込めるかとか，その逆に自分が作成したファイルが他のソフトウェアで利用できるかといったあたりです．

インターネットで利用するソフトウェアなら，たとえばメールならSMTPプロトコルやPOP3プロトコルにしたがっているかとか(独自プロトコルを使った場合，そのプロトコルを理解できる狭い範囲でしか通用しない)，GUIプログラムなら，それが動作する環境の約束ごと(controlキーとCキーでコピーを意味するとか，マウスの右ボタンを押すとポップアップメニューが出るなど)にしたがっているかです．単体で動作するプログラムであっても，ユーザーが無意識のうちに期待する挙動に反していると，使い勝手が悪いと判断されます．これは広い意味での互換性になりますが，ユーザーの期待に反するものは受け入れられにくいものです．

その他の外的品質要因

『オブジェクト指向入門』では，五つの要因以外にも次のような外的品質要因があると指摘しています．

- **効率性：**ハードウェア資源を効率的に利用する能力
- **移植性**[注3]**：**さまざまなハードウェア環境/ソフトウェア環境への移植のしやすさ
- **実証性：**検査の準備や実行，検証のやりやすさ
- **統合性：**不当なアクセス，修正から構成部分(プログラム，データ，ドキュメント)を守る能力
- **使いやすさ：**ソフトウェアシステムの使い方の習得のしやすさ，誤動作やエラーからの回復のしやすさなど

いずれにせよ外的品質要因はいろいろなものがありますが，それぞれの要因を良くすることで，全体の品質が上がります．しかし“トレードオフ”というものもあります．移植性を良くしようとすると効率性が悪くなり，その反対に効率性を良くしようとすると特定の環境では最速を誇るものの他の環境への移植性を悪くしたり，他の環境での効率性を悪くするようなことです．品質の向上は，耳障りは良いのですが厳密に適用しようとすると，とたんに開発日程や費用が高くつくハメになり，どこかで落としどころを考えないと，ちっとも先に進めなくなることもあるので注意したいものです．

モジュール性

『オブジェクト指向入門』の第2章では，モジュール性を論じています．そして，この章で「開放/閉鎖原則」が述べられています．

ソフトウェアの品質(とくに拡張性，再利用性，互換性)を高めるためには，柔軟なシステムアーキテクチャが必要となります．柔軟にしようとすれば，一枚岩構造ではなく複数のモジュールに分かれた“分割統治”の手法を取らざるをえません．しかし，ここでもモジュールの品質が当然ながら問われます．設計レベル/設計手法でのモジュールの品質に関して五つの基準があると同書は指摘しています．

- **モジュールの分解のしやすさ：**一つの問題を互いに独立した複数の小問題に分解できる能力
- **モジュールの組み合わせやすさ：**まったく別の環境においても自由にソフトウェア要素を組み合わせて新規システムを作ることができる能力
- **モジュールの理解しやすさ**[注4]**：**すべてのモジュールを見なくとも個々のモジュールだけを理解しやすいか
- **モジュールの連続性：**小さな仕様変更を加えた場合，その変更が一つか少数のモジュールにとどまるか
- **モジュールの保護性：**実行中のモジュールで異常が発生した場合，その影響が該当モジュールだけか周辺の少数モジュールだけにとどまるか

● モジュールの分解のしやすさ

分割統治をする目的は，一つの巨大な問題を分解して複数の小さな問題にすることにより，一つ一つの小さな問題を把握しやすく，取り扱いやすくさせ，問題解決を楽にしたり，一つ一つの小問題を多人数に分配して日程を縮めるところにあります．その逆に，把握しにくく取り扱いにくい小問題に分解され，問題解決が困難になったり，多人数に分配できず日程も縮まらないなら，その設計手法はどこかが間違っているということです．

ただ勘違いしてはならないのは，一つの巨大な問題を分解して複数の問題にしたはよいのだが，それが「複数の大きな問題」になっていた場合です．ここからさらに，もっと複数の小さな

注3：原文はportability．訳書では「携帯性」と訳しているが，文脈から判断すれば移植性のほうが適切であろう．
注4：原文はunderstandability．訳書では「わかりやすさ」となっている．

問題に分解できるなら，分解の作業が足りなかったということです．一つのシステムを分解して複数のサブシステムに分解すると，それぞれのサブシステムもさらに分解できる可能性はあります．古典的な設計手法と思われている「トップダウン設計」は，基本的にはその考えです．分解するにつれサブシステムのツリーができ，さらにサブシステムの下にツリーができていくわけです(**図1**)．

● モジュールの組み合わせやすさ

分解とは逆の方向，すなわち合成での評価基準です．これは再利用性につながる要因でもあります．うまく分解できれば，その逆の合成も楽勝と思うとそれは大間違いで，トップダウン設計で複数のモジュールを作った場合，あとで再利用することを意図して分解したのでないかぎり，単なる分解になり下がり，あとで複数のモジュールを組み合わせて再利用しにくくなります．

その理由は，分解の基準が特定の要求をかなえるため特定の小問題に分解してしまうところにあります．つまりトップダウン設計の対象となる問題に特化して“汎用性”を無視しているからです．しかしながら設計の段階で，どれが特殊な問題なのか，どれが汎用性なのかは切り分けが難しいところでもあります．プログラミングの経験が少ないと切り分け能力が低いのは，誰でも想像がつくところでしょう．しかしながら経験が多いからといって切り分け能力が高いという保証がないのも実状です．

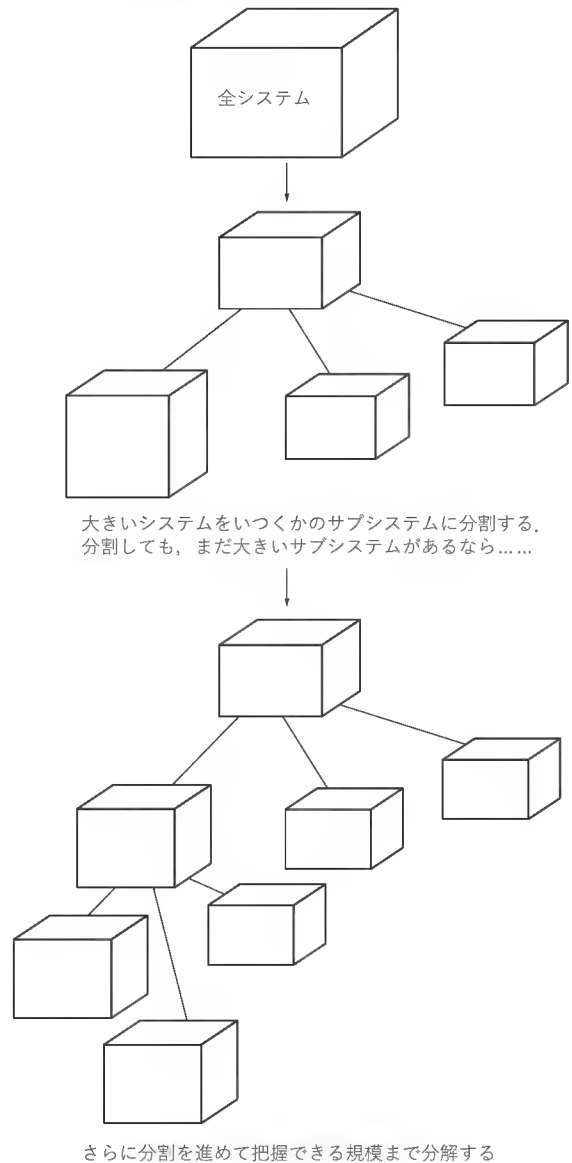
● モジュールの理解しやすさ

すべてのモジュールを見ないと，それぞれのモジュールがどういう働きをするのか理解できないとなると，あとで個々のモジュールを再利用するのが難しくなるだけではなく，保守作業も難しくなります．ドキュメントが不備であるから理解しにくいという理由なら，ドキュメントを整備すれば解決の方向に向かうでしょう．しかしながらモジュール単体やモジュール群が「箱根細工」のようになっていたり，特定の決められた順番や手順で実行されるといった妙な前提条件があると，後から参加した者には容易に理解できず，再利用や保守がしにくいものになります．

● モジュールの連続性

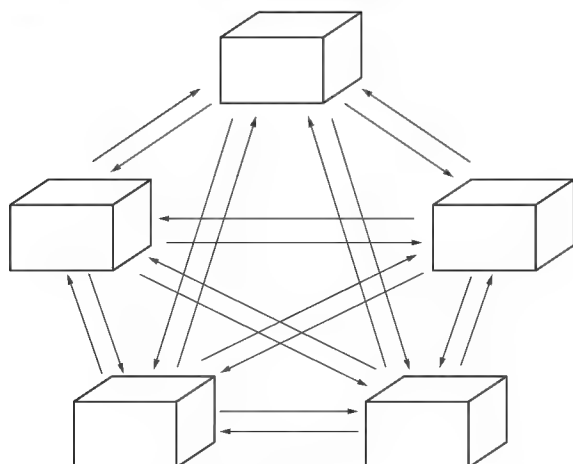
『オブジェクト指向入門』によれば，“連続性”とは解析数学の連続関数の概念から類推して採用したもので，独立変数をわずかに変更した場合，結果として生じる変化も小さいという説明です．うまくモジュールの分解ができているなら，小さな仕様変更程度なら一つか少数のモジュールをいじるだけですむはずです．

ところが，あらゆるモジュールをいじるハメになると，保守がたいへんです．また再利用という点でも怪しいものです．一つの玉を突いただけで他の玉をはじいてしまうようでは，安心して単体のモジュールを取り出せるかということです．同書ではふれていませんが，やたらにグローバル変数を使っていて，たった1箇所を変更しようとしても将棋崩しのように影響が連鎖反応を起こして他のモジュールへ及ぶような作りになっている場合，モジュールの連続性はそこなわれています．もちろん，そ

〔図1〕トップダウン設計によるサブシステムのツリー

のような作りだと，モジュールの組み合わせやすさもそこなわれていることが多いのですが……

● モジュールの保護性

ソフトウェアでは，外部要因のエラー(たとえばハードウェア資源の故障，メモリの枯渇など)はつきもので，その場合，いかに傷口を浅くおさえられるかも品質に関わってきます．ところがこれはあとから検証が難しく，実際に出回っているソフトウェア製品を見ると，かなり怪しいものといわざるをえません．それぞれのモジュールできちんとエラー検出を行い，エラー対策をしていて，エラーの影響を外部モジュールに及ぼさない作りになっているなら保護性は高いでしょう．『オブジェクト指向入門』でも取り上げていますが，プログラミング言語の仕様として「例外」が使える場合，保護性が怪しくなる例が出てきます．同書ではPL/I，CLU，Adaで説明していますが，C++やJavaで

〔図2〕モジュール同士の対話が無節操

モジュール同士の対話が無節操だと，モジュールの連続性と保護性が損なわれやすくなる

〔図3〕1個のボスに集中する構成

1個のボスに集中するMediatorパターンの構成．対話の本数は減るが，ボスの負荷は大きくなる

も似たような状況です．例外の検出と回復処理が別モジュールに分かれ，モジュールの連続性がそこなわれることがあるのも同じです．残念ながら例外を信頼性のあるソフトウェア作りに活用するのでなく"手軽なgoto文"と勘違いした使われ方もたまに見受けます．

モジュール性確保の五つの原則

適切なモジュール性を確保するために五つの原則があると『オブジェクト指向入門』では指摘しています．

- **言語としてのモジュール単位：**モジュールは使用する言語の構文単位に対応していること
- **少ないインターフェース：**モジュールはできるだけ少ないモジュールと対話すること
- **小さいインターフェース（弱い結び付き）：**二つの対話するモジュール同上で交換する情報量はできるだけ少ないこと
- **明示的なインターフェース：**二つのモジュールが対話する場合，その対話があることが明確であること
- **情報隠ぺい：**公開すると決めていないモジュールの情報は非公開になっていること

● 言語としてのモジュール単位

モジュール性を確保するためには，使用するプログラミング言語（あるいはプログラム設計言語，仕様記述言語）がモジュール性をサポートしている必要があります．たとえば古典的なBASIC[注5]ではモジュール構造をサポートしていないため，これを利用してモジュール性を確保することは困難です．

しかし，この意見は批判にさらされることが多く，事情がわかっている設計者やプログラマだけで構成されたコミュニティなら問題がないだろう，どういう道具を使うかではなく，どういう思想を使うかの問題であると反論されることがあります．このあたりは『オブジェクト指向入門』の作者が西欧人流の非常に厳しい見方をしていると感じるところです．しかし，いずれ概念と実装の乖離がひどくなり，保守やレベルアップの段階で耐え難くなると指摘しています．当初の理念がだんだん薄まってしまい，気がついたら誰も当初がどうだったか気にしなくなり，現実に流されるという事態はありがちです．個人レベルでかかえる理念や信念のようなあいまいで思い込みの混じりやすいものではなく，厳密な定義が守れるシステムを構築するほうが先決という考えなのでしょう．

● 少ないインターフェース

これは前回の"デメテルの法則"にも通じる話です．モジュール同上の対話が無節操だと，それだけモジュールの連続性と保護性が損なわれやすくなります（**図2**）．対話を少なくすれば良くなる直接の保証はありませんが，変更やエラーの影響の伝播が減るため，そのぶん有利になります．モジュールの対話を1個のボスに集中することで対話の本数を最小にする構成もあれば（**図3**），隣同上をつなげて特定のボスを作らずに対話の本数を減らす構成もあります（**図4**）．

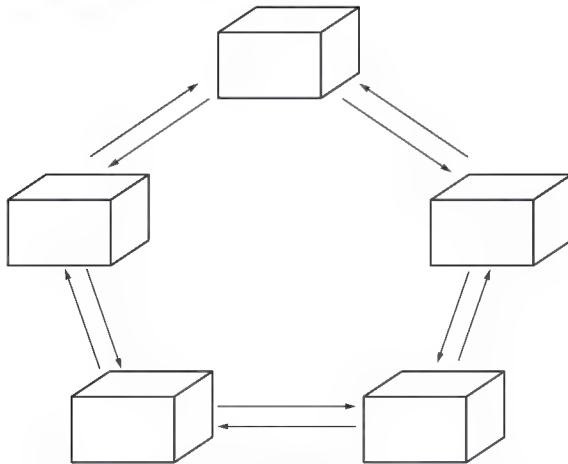
● 小さいインターフェース

「少ないインターフェース」と混同しそうですが，こちらは対話の本数が少ないという意味ではなく，1本の対話内での量的な小ささを意味します．これも連続性と保護性に関わります．

● 明示的なインターフェース

二つのモジュール同上で情報の受け渡しがある場合，プロシージャ呼び出しの引き数，共有変数，スレッド間通信などいろいろ考えられますが，いずれにせよ，どの手段やタイミングで情

注5：ここでいっているのは発明された当初のコンセプトを守っているBASICという意味であって，BASICが古典的だという指摘ではない．最近，利用されているVB.NETやREALbasicあたりだとモジュール性はもちろんのこと，オブジェクト指向プログラミング言語としても十分な機能を有しており，これらを使うならモジュール性は確保できる．

〔図4〕隣同士をつなげる構成

対話の本数が減り連続性と保護性が増すが，自分から隔たったモジュールとの対話は遅れる．また，途中のモジュールが対話を仲介する手間が生じる

報の受け渡しがあるかが明白であり，隠れた影響[注6]がないことです．これはモジュールの理解しやすさ，連続性に関わります．

● 情報隠ぺい

オブジェクト指向の入門記事で必ず強調されるのが情報隠ぺいです．「隠ぺい」というあまり印象の良くない語感からネガティブなことを連想する人もいますが，それは誤解です．オブジェクト指向で強調される隠ぺいとは「プログラミングはインターフェースに沿って行うべきで，実装の中身にしたがうべきでない」という考えがあり，「実装の中身にわずらわされるプログラミングをすべきでない」という考えで出てくるものです．たとえば，人口の男女比を集計するモジュールがあり，

- インターフェース：男の数をカウントアップする，女の数をカウントアップする，男女別と合計の数を得る

というインターフェースを示したとします．男女の数をどう保持するか，つまり実装の中身を，

- 実装A案：男の数を整数型変数でもつ，女の数を整数型変数でもつ
- 実装B案：合計の数を整数型変数でもつ，男の数を整数型変数でもつ
- 実装C案：合計の数を整数型変数でもつ，女の数を整数型変数でもつ

のいずれにするかはモジュールの“実装者”の問題ですが，一方，モジュールの“利用者”にとっては実装の違いは問題になるでしょうか？　利用する側は男の数を得るときに，

- 実装A案，B案：男の数の整数型変数を返す
- 実装C案：「合計の数の整数型変数－女の数の整数型変数＝」で計算した結果を返す

のどちらで実装されようと，たいして関係がないはずです[注7]．むしろ，実装の中身がわかってしまったために，利用者が裏技的にモジュールを利用する影響や，実装の中身を気にしたプログラムによって利用者の開発効率が落ちる影響のほうが問題になるでしょう．楽屋裏の事情を知ることにより悪影響が出るなら，楽屋裏を隠ぺいすることで利用者にはインターフェースのみにプログラミング作業を集中させるほうが得策です．

「開放/閉鎖原則」にいく前に

『オブジェクト指向入門』では，いよいよここから開放/閉鎖原則の話に突入するのですが，ここまでの前フリの話が理解できたでしょうか．理解できなかったとすれば，開放/閉鎖原則の理解以前につまづいています．オブジェクト指向というと，難解な用語で「机上の空論」を述べるという偏見がいまだにありますが，ここまで述べてきたことは，オブジェクト指向以前の時代，つまり構造化プログラムの時代でも，やはり重要な品質評価基準であったり，原則であったものが含まれています．それどころかソフトウェア開発の歴史が始まった時点であっても，やはり真理ではないかと思えてなりません．

次回は，いよいよ開放/閉鎖原則の核心にふれます．お楽しみに．

みやさか・でんと　miyadent@anet.ne.jp

注6：たとえば，ポインタを使ったトリッキーな手法で相手を影響させるなど．

注7：しいて問題が起き得るとしたら，実装方法によってプログラムの実行効率が落ちてしまうのが困るという事態ぐらいだろう．だとしても，それは実装者の責任であり，利用者の落ち度ではない．

TMS320C6711搭載DSPスタータキットとPCM3003搭載オプションボードを使った
ステレオオーディオDSPプログラミング入門(応用編)

三上直樹

今回は，前回説明した基本的なプログラムを使って作成する，多少複雑なプログラムを二つ紹介します．一つは，外部から中心周波数を可変できる帯域通過フィルタで，もう一つは周波数シフタです．

1. 外部から中心周波数を可変できる帯域通過フィルタ

次のような機能をもったフィルタを作成します．右チャネルに入力された信号に対しては，帯域通過フィルタをかけ，それを出力します．このとき，帯域通過フィルタの中心周波数を，左チャネルに加えられた信号の基本周波数でリアルタイムにコントロールできるようにします．

全体は，**図1**に示すように，PLL(phase-locked loop)を使って入力信号の基本周波数を求める部分と，中心周波数を可変できる帯域通過フィルタから構成されます．そこで，最初にPLLと可変帯域通過フィルタについて説明した後に，全体の構成とそのプログラムについて説明していきます．

1.1 PLL

図2には一般的なPLLのブロック図を示します．ディジタル信号処理方式のPLLは，**図2**に示される位相比較器の構成方法により，2種類に分類されます．一つは乗算を使う方法です．この方法は，アナログ方式でよく使われる方法をディジタル信号処理にたんに置き換えたものです．もう一つは，入力信号の位相成分をarctan関数により求め，これとPLL内部のVCOの位相を，減算により比較するという方法です．こちらはディジタル信号処理の特徴を活かした方法ということができます．

位相比較器として乗算を使う方法については，筆者の作ったプログラムがすでに参考文献1)の第8章に載っています．そこで，今回は位相比較器としてarctan関数を使う方法でPLL(以下略してATan-PLL)を作成します．

位相比較器としてarctan関数を使う方法については，参考文献2)に詳しく説明されているので，ここでは簡単に説明します．

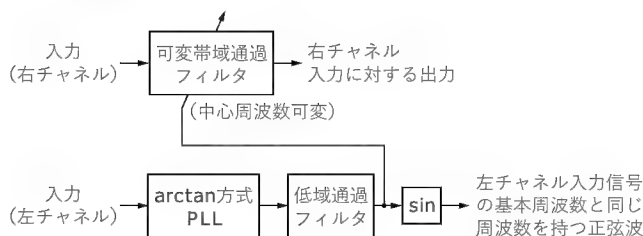
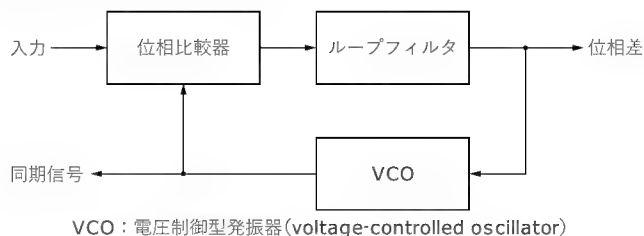
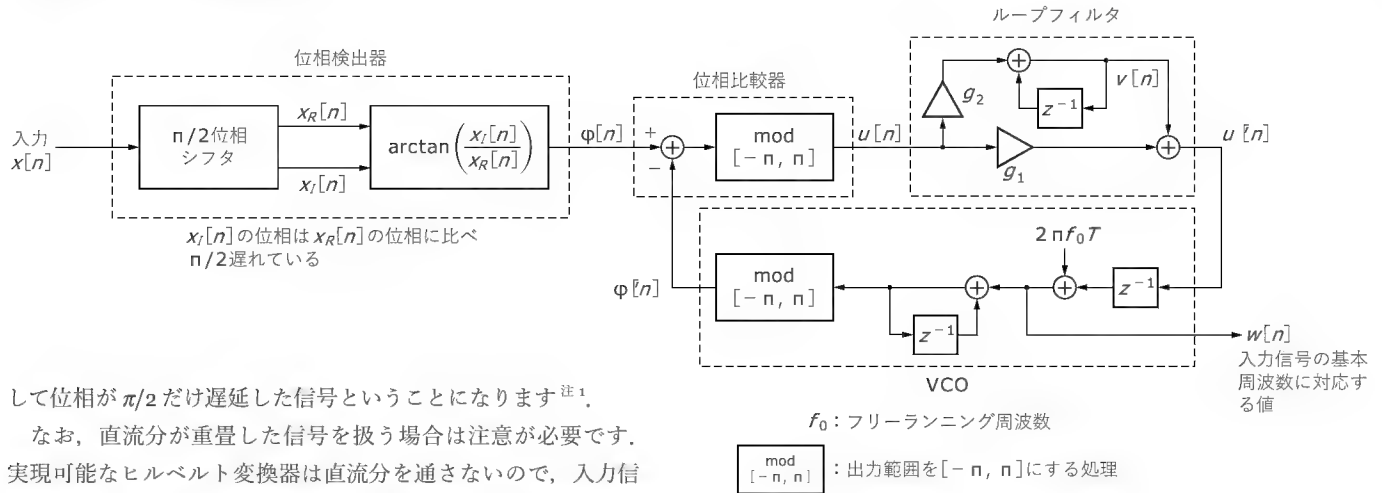
図3にATan-PLLの全体の構成を示します．全体は，位相検出器，位相比較器，ループフィルタ，VCO(Voltage-Controlled Oscillator)の四つの要素から構成されています．以下では，各要素について説明します．

● 位相検出器

位相比較器は，$\pi/2$位相シフタとarctanの計算の，二つの部分から構成されます．

$\pi/2$位相シフタは，入力信号に対して位相が$\pi/2$遅れた信号を生成します．その処理を行うためのブロック図を**図4**に示します．このように，$\pi/2$位相シフタはヒルベルト変換用FIRフィルタ(以下，ヒルベルト変換器と呼ぶ)により実現できます．なお，ヒルベルト変換器については**コラム**(p.130)を参照してください．

このようにして作成したヒルベルト変換器の出力信号$x_I[n]$は入力信号$x[n]$に対して位相が$\pi/2$遅延するだけではなく，このFIRフィルタの次数をMとすると，さらに$MT/2$(T：標本化間隔)の遅延を生じます．一方，**図4**に示される$M/2$段目の遅延器に現れる信号$x_R[n]$は入力信号$x[n]$に対して$MT/2$の遅延を生じます．したがって，$x_I[n]$と$x_R[n]$を以降で使うことにすれば，$MT/2$の遅延はキャンセルされます．つまり，$x_I[n]$は$x_R[n]$に対

〔図1〕外部から中心周波数を可変できる通過帯域フィルタの全体の概略

〔図2〕PLLの基本的な構成

〔図3〕ATan-PLLの全体の構成

して位相が$\pi/2$だけ遅延した信号ということになります[注1].

なお，直流分が重畳した信号を扱う場合は注意が必要です．実現可能なヒルベルト変換器は直流分を通さないので，入力信号に直流分が重畳している場合でも，出力信号$x_I[n]$は直流分を含まないので，問題はありません．一方，$M/2$段目の遅延器に現れる信号$x_R[n]$は入力信号を単に遅延させただけなので，入力信号に直流分が重畳している場合，信号$x_R[n]$は直流分を含むことになります．そうなると，arctan関数を使って位相を正確に求めることはできません．

そこで，入力信号に直流分が重畳している場合には，実際にプログラムを作成する際に，ヒルベルト変換器の前段に直流分除去のためのフィルタを入れる必要があります．このフィルタについてはプログラムの項で説明します．

$\pi/2$位相シフタからの出力信号$x_I[n]$，$x_R[n]$から，入力信号の位相$\phi[n]$はarctan関数を使って，次のように求めることができます．

$$\phi[n]=\arctan\left(\frac{x_I[n]}{x_R[n]}\right) \quad \cdots\cdots(1)$$

この計算をC/C++で標準にサポートされている関数`atan2()`を使う場合，求められる$\phi[n]$の範囲は$[-\pi, \pi]$になります．

● 位相比較器

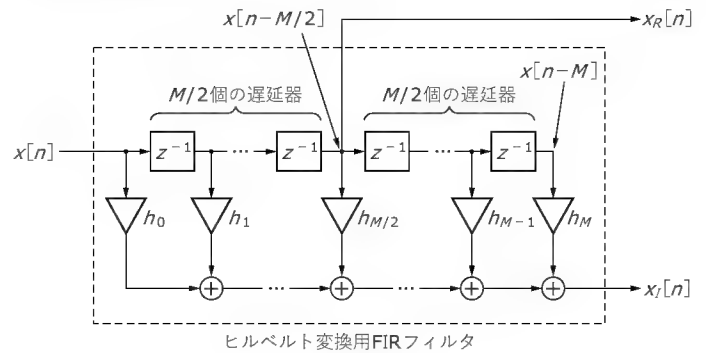
位相比較器では位相検出器の出力とVCOの位相出力を比較します．ここで採用している方式では，単に減算をするだけです．ただし，位相検出器の出力$\phi[n]$とVCOの位相出力$\phi'[n]$の範囲は$[-\pi, \pi]$になるので，そのままでは正しい位相差を求めることができません．そのため$\phi[n]-\phi'[n]$という減算を行った後で，その値の範囲が$[-\pi, \pi]$になるような処理を行います．

● ループフィルタ

位相比較器の出力はループフィルタに与えられます．ループフィルタには完全積分型のIIRフィルタを使います．このフィルタの伝達関数を$H(z)$とすると，

$$H(z)=g_1+\frac{g_2}{1-z^{-1}} \quad \cdots\cdots(2)$$

になります．ループフィルタの入力信号$u[n]$を，出力信号$u'[n]$

〔図4〕$\pi/2$位相シフタのブロック図

をとすると，対応する差分方程式は次のようになります．

$$\begin{cases} v[n]=v[n-1]+g_2u[n] \\ u'[n]=g_1u[n]+v[n] \end{cases} \quad \cdots\cdots(3)$$

このフィルタの係数，はPLLの特性を大きく左右します．係数の決め方については，参考文献2)の1.5.4の項を参考にしてください．

● VCO

通常のVCOの出力は正弦波になります．しかし，ここで実現するATan-PLLの場合，位相比較器の入力は位相そのものになっているので，正弦波を出力する必要はありません．したがって，このVCOは位相を出力することになります．ただし，そのままでは正しい位相差を求められなくなります．そのため，VCOの出力$\phi'[n]$の範囲が$[-\pi, \pi]$になるような処理を行ってから出力します．

図3のVCO部に示されるf_0はフリーランニング周波数です．したがって，VCOの入力が0の場合，f_0がVCOの出力の周波数になります．また，このVCO部の途中から，$w[n]$という信号を取り出していますが，PLLが入力信号に同期している場合，

注1：一般に，信号$\cos[2\pi fn]$に対して，$\cos[2\pi fn]+j\sin[2\pi fn]$という信号は解析信号(analytic signal)と呼ばれる．ところで，$\sin[2\pi fn]$は$\cos[2\pi fn]$に対して，位相が$\pi/2$遅延した信号と考えることができるので，解析信号の虚部は実部に対して位相が$\pi/2$だけ遅延していることになる．以上のことを考慮して，遅延していない信号は実部(real part)に相当するので$x_R[n]$，位相が$\pi/2$遅延した信号は虚部(imaginary part)に相当するので$x_I[n]$という表現を使っている．

ヒルベルト変換器

ヒルベルト変換器(Hilbert transformer)とは，どんな周波数の入力信号に対しても，位相が$\pi/2$遅れた信号を出力するものです．実際には，そのようなものは実現できず，近似として実現することになります．しかし，最初は理想的なヒルベルト変換器から説明していきます．また，ここではディジタル信号処理を考えているので，このコラムでは頭に"離散的"という修飾語を付けることにします[注A]．

理想的な離散的ヒルベルト変換器の周波数特性$H(\omega)$は，式(A)のように与えられます[4]．ただし，ωは正規化された角周波数とします．つまり，標本化角周波数を2πと考えることにします．

$$H(\omega)=\begin{cases}-j, & 0\le\omega<\pi\\ j, & -\pi\le\omega<0\end{cases} \quad \cdots\cdots (A)$$

これに対するインパルス応答が，離散的ヒルベルト変換器の係数そのものに一致します．インパルス応答$h[n]$は，$H(\omega)$の逆フーリエ変換[注B]で与えられるので，以下のようになります．

$$h[n]=\frac{1}{2\pi}\int_{-\pi}^{\pi}H(\omega)e^{j\omega n}d\omega = \begin{cases}0, & n：偶数\\ \dfrac{2}{n\pi}, & n：奇数\end{cases} \quad \cdots\cdots (B)$$

このインパルス応答を**図A**に示します．このように，理想的な離散的ヒルベルト変換器のインパルス応答は，nが偶数の場合に係数の値が0になるという特徴を持っています．また，中心に対して奇対称(点対称)になるという特徴ももっています[注C]．

このインパルス応答はnについて＋側および－側に無限に続くため，そのままではフィルタの係数として使うことはできません．そこで，$h[n]$の$|n|\leqq L$の部分だけを使い，$|n|>L$に対しては0にします．こうすることにより，離散的ヒルベルト変換は，$h[n]$の$|n|\leqq L$の部分を係数とするFIRフィルタにより近似的に実現できることになります．

ただし，リアルタイム処理を行う場合に実現可能なFIRフィルタの係数は，$0<n$に対して0でなければなりません．そこで，$h[n]$に対して$n-L\rightarrow n$という置き換えを行います．つまり，

$$h[-L]\rightarrow h[0],\ h[-L+1]\rightarrow h[1],\ \cdots\cdots,\ h[L]\rightarrow h[2L]$$

となります．

しかし，このままでは，実現されたFIRフィルタの通過域には大きなリプルが生じるので，それを防ぐための何らかの対策が必要になります．それにはいくつかの方法があります．一つの方法は，FIRフィルタの設計でも使われる窓掛けを行うという方法[5]です．

その他に，ヒルベルト変換器の係数を求めるためによく使われているのは，Parks-McClellanによるアルゴリズムを使う方法です．参考文献1)に付属のCDに収録されている筆者が作成したFIRフィルタの設計プログラムは，ヒルベルト変換器の設計もサポートしています．このプログラムはParks-McClellanによるアルゴリズムに基づいて作成したものです．このアルゴリズムを使うと，通過域および阻止域の振幅特性がともに等リプル特性のFIRフィルタを設計できます．

なお，Parks-McClellanによるアルゴリズム使ってヒルベルト変換器を設計した場合，式(B)に示すような，中心に対して係数が奇対称(点対称)になるという特徴は保持されます．しかし，係数の値が一つおきに0になるという特徴は失われます．

〔図A〕理想的な離散的ヒルベルト変換のインパルス応答

注A：本文中ではとくに"離散的"という修飾語は付けない．
注B：正確には離散時間逆フーリエ変換(inverse discrete-time Fourier transform)である．ただし，離散的逆フーリエ変換(inverse discrete Fourier transform)ではない．
注C：理想的なヒルベルト変換器に限らず，近似的なヒルベルト変換器でもこの特徴をもっている．

この信号は入力信号の基本周波数に対応する値になります．入力信号の基本周波数をF_0，標本化間隔をTとすると，次の関係が成り立ちます．

$$F_0=\frac{w[n]}{2\pi T} \quad \cdots\cdots (4)$$

1.2 中心周波数可変帯域通過フィルタ

中心周波数を可変できるフィルタとしては**図5**のブロック図に示すIIRフィルタを使います．このフィルタの伝達関数は，次のようになります．

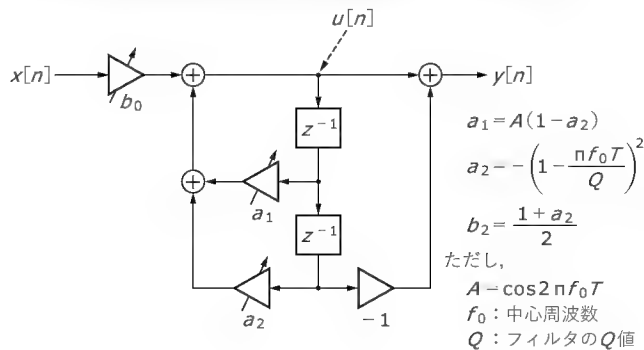
$$H(z)=\frac{b_0\left(1-z^{-2}\right)}{1-a_1z^{-1}-a_2z^{-2}} \quad \cdots\cdots (5)$$

ここでは，フィルタのQ値を一定とし，中心周波数だけを可変できるようにします．このとき，このフィルタの係数a_1，a_2，b_0は以下のようになります．

中心周波数をf_0，標本化間隔をTとすると，a_2が1より小さくて1に近い値の場合，このフィルタのQ値は次のように近似できます[3]．

$$Q\cong\frac{\pi f_0 T}{1-\sqrt{-a_2}} \quad \cdots\cdots (6)$$

〔図5〕中心周波数を可変できる帯域通過フィルタのブロック図

〔図6〕可変帯域通過フィルタの振幅特性

したがって，この式より，a_2は次のようになります．

$$a_2 = -\left(1 - \frac{\pi f_0 T}{Q}\right)^2 \quad \cdots\cdots (7)$$

このa_2を使うと，b_0は次のように表すことができます．

$$b_0 = \frac{1 + a_2}{2} \quad \cdots\cdots (8)$$

次に，定数Aを次のように置きます．

$$A = \cos 2\pi f_0 T \quad \cdots\cdots (9)$$

このAとa_2より，a_1は次のようになります．

$$a_1 = A(1 - a_2)$$

このフィルタの差分方程式は，入力信号を$x[n]$，出力信号を$y[n]$とすると，次に示す式になります．

$$\begin{cases} u[n] = a_1 u[n-1] + a_2 u[n-2] + b_0 x[n] \\ y[n] = u[n] - u[n-2] \end{cases} \quad \cdots\cdots (9)$$

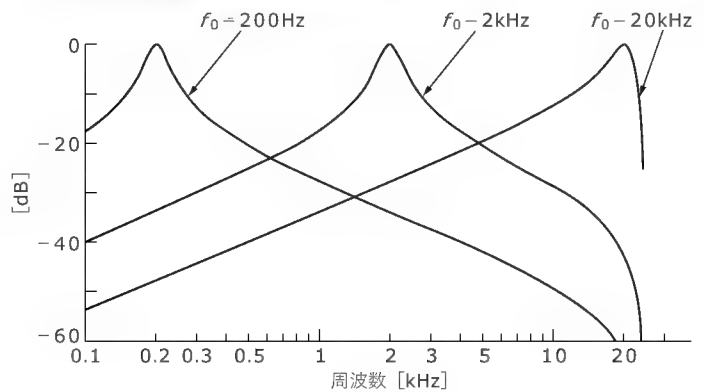
図6には，この可変帯域通過フィルタの振幅特性を示します．この図は，標本化周波数が48kHzで，$Q = 5$とし，中心周波数f_0を200Hz，2kHz，20kHzとした場合の振幅特性です．

1.3　全体の構成

可変帯域通過フィルタの中心周波数は，**図3**のVCO部の途中から引き出された出力$w[n]$から計算します．ただし，$w[n]$は微小に変動しているので，ここでいったん低域通過フィルタを通します．このフィルタの入力を$w[n]$，出力を$w'[n]$とすると，差分方程式は次のようになります．

$$w'[n] = aw'[n-1] + (1 - a)w[n] \quad \cdots\cdots (10)$$

このようにして求めた$w'[n]$から，この値が決められた範囲内になるように処理を行った後，帯域通過フィルタの中心周波数を計算します．

また，$w'[n]$を使い，右チャネルに入力された信号の基本周波数と同じ周波数を持つ正弦波を出力するようにします．これはPLLの同期状態をモニタするために設けます．

1.4　外部から中心周波数を可変できる帯域通過フィルタのプログラム

全体はHilbert.cpp，Coefficients.cpp，VariableBPF_PLL.cppの三つのファイルに分けて作成しました．

● Hilbert.cpp

リスト1にHilbert.cppを示します．これはヒルベルト変換による$\pi/2$位相シフタの部分で，クラスとして実現しています．このクラスHilbertは次の周波数シフタでも使います．

コンストラクタでは，最初にDSKボードの3個のLED（user_LED1，2，3）を点灯します．次に，private部で宣言されている二つのポインタ*un，*snに対して領域を確保し，それぞれの値を0に初期化しています．これらはフィルタの信号を入れておく遅延器に相当します．コンストラクタの引き数はヒルベルト変換で使うFIRフィルタの次数になっています．

最後にDSKボードの3個のLEDを消灯します．これにより，領域が確保できたことを確かめることができます．

領域確保は演算子newで行っています．このとき注意しなければならない点があります．newで確保した領域はヒープ（heap）領域に配置されますが，この大きさはリンカコマンドファイルの“-heap”オプションで決定されます[注2]．したがって，実際に使用する領域の大きさに応じて，“-heap”オプションの数値を設定しておく必要があります．1のプログラムの場合は，前回の**リスト3**に示すリンカコマンドファイルでの指定，つまり“-heap 0x400”[注3]という設定で問題ありませんが，2節のプログラムの場合は，これでは不足します．これについては2節で説明します．

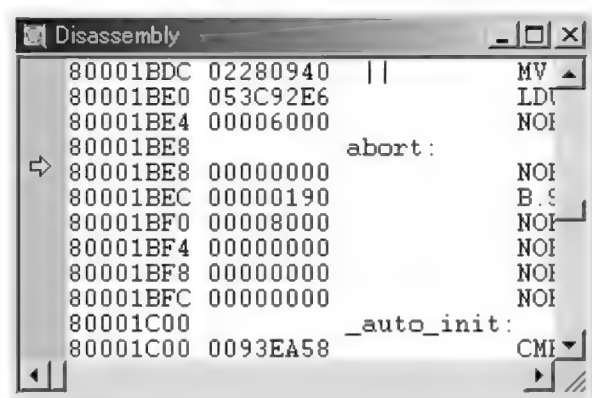
なお，newで領域の確保に失敗した場合は，関数abort()でプログラムを終了するようにしています．したがって，DSKボードの3個のLEDは点灯したままになります．これにより，領域確保が失敗したことがわかります．

注2：ヒープ領域の大きさは，ビルドオプションの中の“Linker”に関するオプションのところでも指定できる．ただし，リンカコマンドファイルで“-heap”オプションによる指定がある場合は，こちらのほうが優先される．

注3：単位はバイト数．

〔リスト 1〕ヒルベルト変換のためのクラス(Hilbert.cpp)

```
//--------------------------------
// class for Hilbert transformer
//--------------------------------
#include <cstdlib>
using namespace std;

class Hilbert
{
private:
    const int order;
    float *un, *sn;
public:
    Hilbert(int set_order);
    ~Hilbert() { delete[] un; delete[] sn; }
    void Execute(const float hn[], const float ab[], float xin,
                 float &x_real, float &x_imag);
};

// constructor of Hilbert
Hilbert::Hilbert(int set_order) : order(set_order)
{
    *(u_v_int *)IO_PORT = 0x00000000;       // turn on LEDs

    if ((un = new float[order+1]) == NULL) abort();
    if ((sn = new float[2]) == NULL) abort();

    for (int n=0; n<=order; n++) un[n] = 0.0;
    for (int n=0; n< 2; n++) sn[n] = 0.0;
    *(u_v_int *)IO_PORT = 0x07000000;       // turn off LEDs
}

// filtering by Hilbert transformer
//      hn      : coefficients for Hilbert transformer
//      ab      : coefficients for DC rejection filter: a0, a1, b0, b1
//      xin     : input
//      x_real  : pi/2-advanced signal compared with x_imag
//      x_imag  : pi/2-delayed signal compared with x_real
void Hilbert::Execute(const float hn[], const float ab[], float xin,
                      float &x_real, float &x_imag)
{
    float acc, yn;

// DC rejection filter of 2nd order (b2 = b0)
    acc = ab[0]*sn[0] + ab[1]*sn[1] + xin;
    yn = ab[2]*(acc + sn[1]) + ab[3]*sn[0];
    sn[1] = sn[0];
    sn[0] = acc;

// input real signal to analytic signal
    un[0] = yn;
    acc = 0.0;
    for (int k=0; k<=order; k++) acc = acc + hn[k]*un[k];
    x_real = un[order/2];   // pi/2-advanced signal compared with x_imag
    x_imag = acc;           // pi/2-delayed signal compared with x_real
    for (int k=order; k>0; k--) un[k] = un[k-1];
}
```

〔図 7〕関数 abort() により終了したときに現れる Disassembly ウィンドウの例

関数 abort() で終了した場合は，Disassembly ウィンドウがアクティブになります．そのときの Disassembly ウィンドウに表示される内容の例を**図 7** に示します．**図 7** からわかるように，abort というラベルの付近で止まっているので，関数 abort() でプログラムが終了したことがわかります．

デストラクタでは，コンストラクタで確保した領域を解放します．

ヒルベルト変換器による $\pi/2$ の位相シフトを行うためのメンバ関数 Execute() は，次のように宣言されています．

```
void Hilbert::Execute(const float hn[],
                  const float ab[],  float xin,
                  float &x_real,  float &x_imag);
```

hn[]	ヒルベルト変換用 FIR フィルタの係数
ab[]	直流分除去フィルタの係数
xin	入力信号
x_real	x_imag に対して位相が進んだ信号
x_imag	x_real に対して位相が遅れた信号

直流分除去フィルタとしては，入力信号を $x[n]$，出力信号を $y[n]$ とすると，次の差分方程式で表される 2 次の IIR フィルタを使いました．

$$\begin{cases} u[n] = a_1 u[n-1] + a_2 u[n-2] + b_0 x[n] \\ y[n] = b_0\left(u[n] + u[n-2]\right) + b_1 u[n-1] \end{cases} \quad \cdots\cdots(11)$$

このフィルタのブロック図を**図 8** に示します．配列 ab[] の内容と係数の関係は**図 8** を見てください．

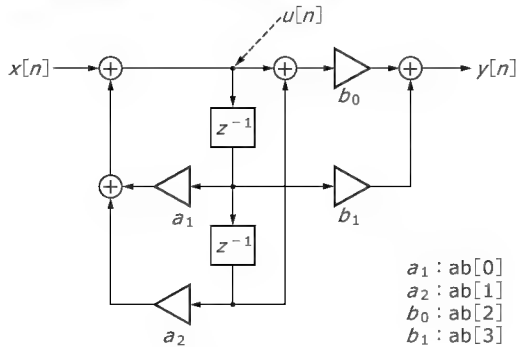
● Coefficients.cpp

Coefficients.cpp は $\pi/2$ 位相シフタで使う，ヒルベルト変換用 FIR フィルタと直流分除去用 IIR フィルタの係数で，これを**リスト 2** に示します[注4]．これらの係数は，参考文献 1) に付属する CD に収録されているディジタルフィルタ設計用プログラムを使って求めたものです．これらを設計したとき与えたパラメータを**表 1**，**表 2** に示します．

● VariableBPF_PLL.cpp

VariableBPF_PLL.cpp は main() 関数を含む全体で，これを**リスト 3** に示します．このプログラムでは標準でサポートされている算術関数を使います．そこで，“cmath”をインクルードします．また，この場合，次の文により名前空間の宣言を行う必要があります[注5]．

注 4：このリストでは，係数の一部が省略されている．係数全体については，InterGiga No.30 に収録予定のソースファイルを参照のこと．
注 5：この宣言は必ずしも行う必要はない．しかし，その場合は，インクルードファイル cmath に含まれる関数を使う場合，関数名の先頭に“ std:: ”を付け加える必要がある．たとえば，std::sin() のように記述する必要がある．

〔図8〕直流分除去フィルタのブロック図

〔表1〕ヒルベルト変換用FIRフィルタの設計パラメータ

項　目	値など
標本化周波数	48kHz
下側帯域端断周波数	0.2kHz
上側帯域端断周波数	23.8kHz
次数	240次

〔表2〕直流分除去用FIRフィルタの設計パラメータ

項　目	値など
標本化周波数	48kHz
遮断周波数	0.1kHz
通過帯域	高域通過
特性	バタワース特性
次数	2次

〔リスト2〕ヒルベルト変換で使うフィルタの係数(Coefficients.cpp)

```
//-------------------------------------------------
// Coefficients for Hilbert transformer
//
//   specification of Hilbert transform filter
//      order              : 240
//      lower band edge    :   0.2 kHz
//      upper band edge    :  23.8 kHz
//      ripple in passband:   0.15257756 dB
//   specification of DC rejection IIR filter
//      order              :   2
//      cutoff frequency   :   0.1 kHz
//      type               : Butterworth
//-------------------------------------------------
//coefficients for Hilbert transformer
const int ORDER = 240;
const float hn[ORDER+1] = {
        -7.23441500e-03, -4.07912338e-03,  2.00523030e-03, -2.96075832e-03,
         1.07803495e-03, -2.37121559e-03,  5.49900033e-04, -2.08430314e-03,
         2.47382526e-04, -1.97672737e-03,  7.24287656e-05, -1.97030745e-03,
   ～中略～
        -2.47382526e-04,  2.08430314e-03, -5.49900033e-04,  2.37121559e-03,
        -1.07803495e-03,  2.96075832e-03, -2.00523030e-03,  4.07912338e-03,
         7.23441500e-03};

// coefficients for DC rejection filter: a1, a2, b0 (=b2), b1
const float ab[4] = {
         1.98148851e+00, -9.81658283e-01,  9.90786698e-01, -1.98157340e+00};
```

```
using namespace std;
```

PLL部は関数PLL()，可変帯域通過フィルタ部は関数VariableBPF()がそれぞれ対応します．また，インライン関数mod2PI()は，引き数の範囲を$[-\pi, \pi]$になるように処理した結果を戻り値とします．

関数PLL()は1.1の項で説明した処理を行います．ループフィルタの係数g1，g2の値は参考文献2)を参考に決めました．この関数PLL()は，**図3**の$w[n]$に対応する値を第2引き数に出力します．この値は，左チャネルからの入力信号の基本周波数に対応する値ですが，そのままでは変動が激しい場合があるので，式(10)に示す低域通過フィルタを通します．この値に対して，範囲が$[-\pi, \pi]$になるように処理をします．この値を関数sinf()とVariableBPF()に与えます．

sinf()で計算した値は，左チャネルから出力し，PLLの同期がうまく取れているかどうかのモニタとして使います．

一方，VariableBPF()ではこの与えられた値に対して中心周波数に対応する値を求め，それに基づいて帯域通過フィルタの係数を計算します．Q値は5とします．

その後，帯域通過フィルタの処理を行います．

● ビルドについて

ビルドする際は，前回紹介したリセットベクタvect_Reset.asm(**リスト2**)，リンカコマンドファイルlnk_std.cmd(**リスト3**)をそのまま使います．また筆者はリリース用の設定でビルドを行っています．

2. 周波数シフタ

1.1の項で説明した$\pi/2$位相シフタを使うと，周波数シフタを簡単に実現することができます．周波数シフタとは，入力された信号の周波数f_0に対して，ある周波数f_1だけ周波数を移動するシステムです．ここでは，出力の周波数がf_0+f_1になるようなシステムを作成します．

2.1 周波数シフタの原理

まず，二つの正弦波[注6]を考え，それぞれの周波数をf_0，f_1とすると，$\cos[2\pi f_0 n]$，$\cos[2\pi f_1 n]$と書くことができます．これらの正弦波に対して位相が$\pi/2$だけ遅延した信号は$\sin[2\pi f_0 n]$，$\sin[2\pi f_1 n]$となります．そこで，これらに対して次のような信号を考えます．

$$\cos[2\pi f_0 n]+j\sin[2\pi f_0 n],\quad \cos[2\pi f_1 n]+j\sin[2\pi f_1 n] \quad \cdots\cdots\cdots\cdots(12)$$

この式で，jは虚数単位を表します．式(11)で表現される信号は，$\pi/2$位相シフタの出力と考えることができます．

式(12)はオイラーの公式[注7]を使うと，次のように複素指数関数で表すことができます．

$$\begin{aligned}\exp[j2\pi f_0 n]&=\cos[2\pi f_0 n]+j\sin[2\pi f_0 n]\\ \exp[j2\pi f_1 n]&=\cos[2\pi f_1 n]+j\sin[2\pi f_1 n]\end{aligned} \quad \cdots\cdots\cdots\cdots(13)$$

この二つの複素指数関数の積は次のようになります．

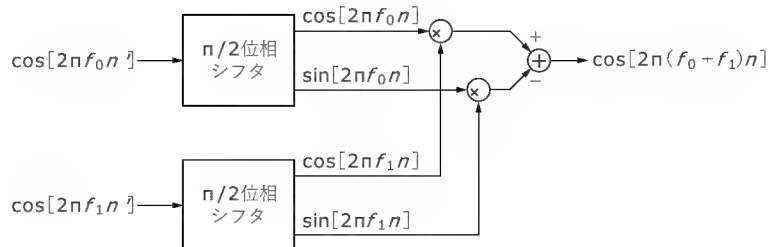
注6：ここではsinとcosの違いは本質的ではないので，$\cos[2\pi f_0 n]$なども正弦波と呼ぶことにする．
注7：オイラーの公式：$\exp(jx)=\cos x+j\sin x$，$\exp(-jx)=\cos x-j\sin x$

〔リスト3〕外部から中心周波数を可変できる通過帯域フィルタ(VariableBPF_PLL.cpp)

```
//---------------------------------------------------------
//  Variable BPF controlled by the fundamental frequency of
//      the signal driven from left channel.
//  The fundamental frequency is extracted by PLL.
//
//      right channel : signal to be filterd
//      left channel  : control signal of center frequency
//---------------------------------------------------------
#include <cmath>
#include "PCM3003 Polling.cpp"
#include "Hilbert.cpp"
#include "Coefficients.cpp"
using namespace std;

const float Ts   = 1.0/48000;           // sampling period
const float PIf  = 3.141592654;
const float PI2f = 6.283185307;

PCM3003 codec;
Hilbert filter(ORDER);

inline float mod2PI(volatile float x)
{
    if (x >  PIf) x = x - PI2f;
    if (x < -PIf) x = x + PI2f;
    return x;
}

inline float square(float x) { return x*x; }

void PLL(float xn, float &wn);
void VariableBFP(float xn, float &yn, float f0Ts);

int main()
{
    const float fLTs = 200.0*Ts;        // lower limit of center frequency
    const float fHTs = 2.0e4*Ts;        // upper limit of center frequency
    const float PI2inv f = 1.0/PI2f;    // inverse of 2*pi
    const double aLPF = 0.999;          // coeffisient of smoothing filter
    const double bLPF = 1.0 - aLPF;     // coeffisient of smoothing filter

    float ch0, ch1, wn, f0Ts, phase;
    double yLPF;

    phase = 0.0;
    yLPF = 0.0;

    while (1)                           // endless loop
    {
        codec.ReadRdy(ch0, ch1);

//-------------------------------
//      left channel
//-------------------------------
        PLL(ch1, wn);
// Convert phase to sin sinusoidal wave
        yLPF = aLPF*yLPF + bLPF*wn;     // LPF, must be use double type
        phase = mod2PI(phase + yLPF);
        ch1 = 0.5f*sinf(phase);         // output of VCO

//-------------------------------
//      right channel
//-------------------------------
        f0Ts = yLPF*PI2inv f;
        if (f0Ts > fHTs) f0Ts = fHTs;
        if (f0Ts < fLTs) f0Ts = fLTs;
        VariableBFP(ch0, ch0, f0Ts);

        codec.Write(ch0, ch1);
    }
}

//--------------------------------------------
// PLL using linear phase comparator
//--------------------------------------------
void PLL(float xn, float &wn)
{
    const float g1  = 0.4;
    const float g2  = 0.05;
    const float w0  = PI2f*1000.0*Ts;   // free-running freq. of PLL: 1000 Hz

    float xr, xi, un, ph, udn;
```

〔リスト3〕外部から中心周波数を可変できる通過帯域フィルタ(VariableBPF_PLL.cpp)(つづき)

```
    static float vn = 0.0;
    static float phd = 0.0;

    filter.Execute(hn, ab, xn, xr, xi);
// get phase of input signal
    ph = ( (xr==0.0) && (xi==0.0) ) ? 0.0 : atan2f(xi, xr);
    un = mod2PI(ph - phd);               // compare phase

// loop filtering
    vn = vn + g2*un;
    udn = g1*un + vn;                    // udn: output of loop filter

// VCO
    wn = udn + w0;                       // wn/T : angular frequency of center
    phd = mod2PI(phd + wn);              // phd: output of VCO
}

//-------------------------------------------------------
//  Variable 2nd order BPF with zeros at z = 1 and -1
//      right channel : signal to be filterd
//      left channel  : control center frequency of BPF
//-------------------------------------------------------
//      xn   : input signal
//      yn   : output signal
//      f0Ts : center frequency * sampling period
void VariableBFP(float xn, float &yn, float f0Ts)
{
    const float Qinv = 1.0/5.0;          // inverse of Q
    float a1, a2, A, b0, un;
    static float un1 = 0.0;
    static float un2 = 0.0;

    A = cosf(PI2f*f0Ts);
    a2 = -square(1.0f - PIf*f0Ts*Qinv);
    a1 = A*(1.0f - a2);
    b0 = 0.5f*(1.0f + a2);

    un = a1*un1 + a2*un2 + b0*xn;
    yn = un - un2;
    un2 = un1;                           // shift data
    un1 = un;                            // shift data
}
```

〔図9〕$\pi/2$位相シフタによる周波数シフタの構成

$$\exp[j2\pi f_0 n] \times \exp[j2\pi f_1 n] = \exp[j2\pi(f_0 + f_1)n] \quad \cdots\cdots\cdots\cdots(14)$$

この式にオイラーの公式を適用すると，次のようになります．

$$\exp[j2\pi(f_0 + f_1)n] = \cos[2\pi(f_0 + f_1)n] + j\sin[j2\pi(f_0 + f_1)n] \quad \cdots\cdots\cdots(15)$$

この式から実部のみを取り出せば，$\cos[2\pi(f_0 + f_1)n]$という信号が得られます．複素数の実部を$\mathrm{Re}\{\ \}$で表すと，次のようになります．

$$\cos\left[2\pi\left(f_0 + f_1\right)n\right] = \mathrm{Re}\left\{\exp\left[j2\pi\left(f_0 + f_1\right)n\right]\right\}$$
$$= \cos\left[2\pi f_0 n\right]\cdot\cos\left[2\pi f_1 n\right] - \sin\left[2\pi f_0 n\right]\cdot\sin\left[2\pi f_1 n\right] \quad \cdots(16)$$

以上のことから，周波数シフタの処理は**図9**のようになります．

2.2 周波数シフタのプログラム

1.3で作成したHilbert.cpp，Coefficients.cppはそのまま使います．ここでは新たにFrequencyShifter.cppを作成しました．

- FrequencyShifter.cpp

FrequencyShifter.cppはmain()関数を含む全体で，これを**リスト4**に示します．このプログラムは右チャネルから入力された信号の周波数を，左チャネルから入力された信号の周波数だけ高域側にシフトし，これを右チャネルに出力します．左チャネルには右チャネルの入力信号をそのまま出力します．

プログラムは**リスト4**からわかるように，非常に簡単なものです．まず，右左それぞれのチャネルからの入力信号に対し，ク

〔リスト4〕周波数シフタ(FrequencyShifter.cpp)

```
//----------------------------------------------------------
//  Frequency shifter using complex multiplication
//
//  <input>
//      right channel : signal to be frequency-shifted
//      left channel  : control shifting frequency
//  <output>
//      right channel : frequency-shifted signal
//      left channel  : input signal from right channel
//----------------------------------------------------------
#include "PCM3003 Polling.cpp"
#include "Hilbert.cpp"
#include "Coefficients.cpp"

PCM3003 codec;
Hilbert filter0(ORDER), filter1(ORDER);

int main()
{
    float ch0, ch1, xr0, xr1, xi0, xi1;

    while (1)                       // endless loop
    {
        codec.ReadRdy(ch0, ch1);

        filter0.Execute(hn, ab, ch0, xr0, xi0); // right channel
        filter1.Execute(hn, ab, ch1, xr1, xi1); // left channel

        ch1 = ch0;                  // output: input from right channel
        ch0 = xr0*xr1 - xi0*xi1;    // output: frequency-shifted signal

        codec.Write(ch0, ch1);
    }
}
```

〔写真1〕周波数シフタのプログラムの実行のようす
(画面左上に表示されている周波数は右チャネルのもの．カーソルは左チャネルの周波数測定用にセット)

ラスHilbertのメンバ関数であるExecute()を適用し，位相が$\pi/2$だけ遅延した信号を求めます．次に，式(16)の右辺の操作を行い，周波数が高域側にシフトされた信号を求めます．

● ビルドについて

ビルドする際は，リセットベクタとして前回紹介したvect_Reset.asm(**リスト2**)を使います．しかし，前回紹介したリンカコマンドファイルlnk_std.cmd(**リスト3**)をそのまま使うことはできません．それは，このクラスHilbertのコンストラクタで，演算子newによりヒープ領域に動的に確保される領域の大きさが，0x400(＝1024)バイトを超えてしまうからです．

そこで，まずヒープ領域として使われる大きさを求めます．フィルタの実行の際に，信号を入れておく遅延器に相当する配列は，型がfloat型で，大きさが243(＝241＋2)になります．float型は4バイトで表現されるので，243×4＝972バイトになります．このプログラムでは，filter0，filter1の二つのオブジェクトが宣言されているので，全体で1944バイトのヒープ領域が使われることになります．

したがって，リンカコマンドファイルで確保するヒープ領域の大きさは1944バイト以上にする必要があります．そこで，筆者は多少の余裕を持たせるため，リンカコマンドファイルでは，ヒープ領域の大きさの指定を“-heap 0x800”のように記述しています．これで，ヒープ領域の大きさを2048バイトに設定したことになります．

なお，1節の場合と同様に，筆者はリリース用の設定でビルドを行っています．

● 実行結果

このプログラムを実行したときの波形を**写真1**に示します．これは，右チャネルに2kHzの正弦波，左チャネルに500Hzの正弦波をそれぞれ入力した場合です．写真上は右チャネルの出力で，その周波数は画面左上の表示からわかるように2.5kHzになっており，周波数が500Hz高域側へシフトしたことがわかります．写真下は右チャネルに入力された信号をそのまま左チャネルへ出力したものです．写真のカーソルは下の波形の周波数を測定するように設定しています．したがって，画面で上の中央付近の数値は，右チャネルの入力信号の周波数である2kHzを示しています．

参考文献

1) 三上直樹，『C言語によるディジタル信号処理入門』，CQ出版(株)，2002年
2) 萩原将文他 編著，『実用PLLシンセサイザ』，第1章，総合電子出版社，1995年
3) 武部 幹，『ディジタルフィルタの設計』，p.105，東海大学出版会，1986年
4) A. V. Oppenheim, R. W. Shafer, *Digital Signal Processing*, p.359, Prentice-Hall, 1975
5) 三上直樹，『ディジタル信号処理の基礎』，第4章，CQ出版(株)，1998年

みかみ・なおき 職業能力開発総合大学校 情報工学科

インターネットやディジタルカメラなどに使われている静止画像圧縮規格JPEGの新版である「JPEG2000」については，本誌でも何度か解説している．今回は，DSPを使ったJPEG2000デコーダの実装について解説する．一般的に，JPEG2000の符号化アルゴリズムは，分岐命令を苦手とするDSPには向かないと考えられているが，さまざまな工夫をこらすことでDSPに最適化された実装も可能だということを示す．（編集部）

はじめに

JPEG2000は，静止画像圧縮技術の新しい規格として，その技術に関する基本方式を定義しているPart1が2001年1月にIS（International Standard）化され，これまでにディジタルカメラやインターネットなどで普及してきたJPEGに続く技術として大いに期待されています．本誌でも稿末の参考文献に示すとおり，何度か解説されています．

本稿では，JPEG2000の基本方式の概要について，既存の技術であるJPEGの基本方式と対比しながら解説し，DSPを用いた画像処理技術の一例として，JPEG2000のデコーダ（復号化器）のDSPへの実装方法について説明します．

なお便宜上，本文中の「JPEG」はJPEG基本方式について，「JPEG2000」はJPEG2000基本方式について言及しているものと理解してください．

JPEG2000の概要

近年のディジタル画像の用途や通信方法の多様化にともない，これまでのJPEGでは対応しきれない場合も出てきたことから，JPEG2000の規格化が進められてきました．ここでは，JPEG2000のおもな特徴とその代表的なアルゴリズムについて簡単に説明します．

● JPEG2000のおもな特徴

JPEG2000では，おもな特徴として，

a）高画質（とくに高圧縮率での圧縮効率が良い）
b）可逆・非可逆圧縮の統合
c）入力画像の多様性
d）ROI（Region Of Interest）
e）階層的符号化
f）高いエラー耐性能力

といった点があげられます．

従来のJPEGでは，JPEG圧縮そのものが非可逆圧縮であり，圧縮によってデータが失われてしまうという点や，圧縮率を高くするとブロックノイズや歪みが目立つようになるという欠点がありました．**図1**に見られるように，1/100のような高圧縮率時には，JPEG2000の画像は多少ぼやけているものの現画像の原型を残しているのに対し，JPEGの画像はブロックノイズが発生し，画質が大きく劣化しているのがよくわかります．a），b）の特徴は，こういった画質面でのJPEGの欠点を補うものです．

また，JPEGの入力画像はRGBの各成分が8ビット/ピクセルのRGBフルカラー画像を想定しており，画像全体に均一に圧縮処理が施されます．そのため，入力画像成分がRGB24ビット（RGB各色8ビット）以外の画像や，画像の中に重要な領域が存

〔図1〕高圧縮率時でのJPEGとJPEG2000の画質比較

（a）原画像

（b）1/100圧縮JPEG2000

（c）1/100圧縮JPEG

在し，部分的に高画質が要求される画像などにはJPEG圧縮が使えませんでした．c)，d)の特徴により，これまでJPEGが敬遠されてきた画像の分野にもJPEG2000で対応できるようになりました．

図2では同じ画像を同じ圧縮率で，そのまま符号化したものと，d)のROI(Region Of Interest)の技術によって画像の中心部分を注目領域に設定して符号化したものを比較しています．ROIとは，注目領域により多くの符号ビットを割り当てることにより，注目領域の画質をまわりの領域よりも良くする技術です．

ROIを使っていない**図2(a)**は全体的に画質が劣化していますが，ROIを中心部分に使用している**図2(b)**はまわりの領域の画質劣化こそ**図2(a)**よりも激しいものの，注目領域に指定した画像の中心部分(顔のあたり)の画質の劣化はあまり見られません．この技術は，医療用画像で患部の画質を保持したまま画像データを圧縮するといった用途などへの利用が期待されています．

従来のインターネットを媒介とした画像の閲覧やディジタルカメラによる画像の撮影のほか，最近では携帯電話を用いた画像の撮影，伝送やDVDプレーヤで画像を閲覧といった形で，画像を扱うデバイスは多様化してきました．また，画像の伝送方法もFDやCDといったメディアからADSL，携帯電話，PHSなど，多様化しています．

e)，f)の特徴は，このような解像度のまったく異なるデバイスから帯域幅やエラー耐性のまったく異なる伝送方法を用いても，画像が閲覧できるようにするための機能といえます．具体的には，e)の階層的符号化により，JPEG2000はスケーラビリティをもつことができます．JPEG2000のもつスケーラビリティとは，JPEG2000コードストリームの一部分を復号化し，コードストリーム全体を復号化したときと比較して，低解像度や低画質の画像を得ることのできる能力を意味します．

例をあげると，**図3**は画質スケーラビリティをもたせたJPEG 2000コードストリームを段階的に復号化したもの，**図4**は解像度スケーラビリティをもたせたものを段階的に復号化したものです．この技術を応用すれば，データの伝送や画像を閲覧する端末の条件によっては，コードストリームのデータの伝送や復号を途中で止めて，時間や通信のコストを節約することもできます．

〔図2〕ROI使用による効果

(**a**) 1/80圧縮 ROIなし

(**b**) 1/80圧縮 ROIあり

● JPEG2000の代表的なアルゴリズム

JPEG2000の符号化プロセスを**図5(a)**に，JPEGの符号化プロセスを**図5(b)**に示します．ここでは後ほどその実装方法について述べる代表的な二つのアルゴリズム，ウェーブレット変換，係数ビットモデリングを中心に，JPEGのアルゴリズムと比較しながら紹介します．

▶DCレベルシフト

基本的にJPEGやJPEG2000といった画像圧縮のアルゴリズムは，その空間的相関を利用して，できるかぎり元のデータを損失することなく数学的な変換を行うことにより，データに冗長性をもたせ，エントロピー符号化を行うことによってデータを圧縮します．つまり，画像データを0またはそれに近い値に変換すればするほど圧縮率が高まることになります．画像データを0付近に集めるための第1段階として，このDCレベルシフトがあります．

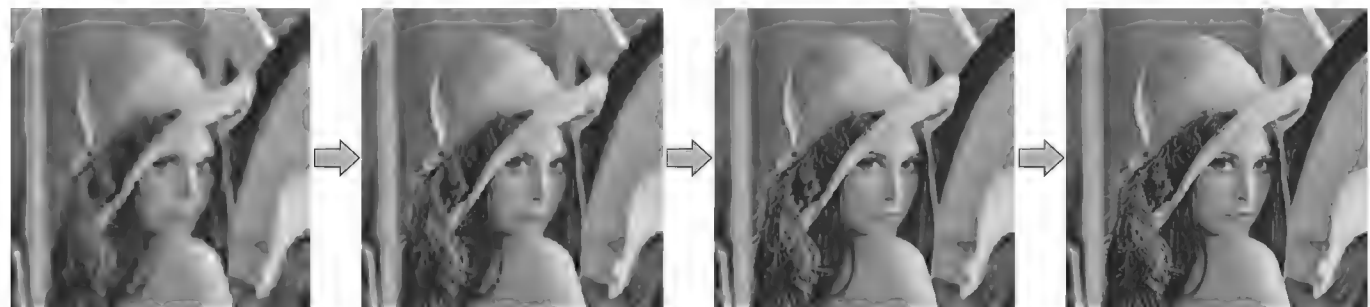
入力信号としてもっとも一般的であるRGB信号は正の整数として表されます．そこで，入力信号の成分のうち，正の値をもっているものにはこのDCレベルシフトを施し，ダイナミックレンジの中心値を0にすることにより，後の符号化効率を向上させます．具体例をあげると，もっとも一般的であるRGB各色8ビットの各サンプルには，式(1)で与えられるレベルシフトが施されます．

$$Y = X - 128 \quad \cdots\cdots(1)$$

ただし，Xは入力信号，Yはレベルシフトした信号

▶色変換

色変換では，同画素内での各色成分の相関から冗長性を生み出します．JPEGでは式(2)で与えられるRGB空間からYC_bC_r空

〔図3〕画質スケーラビリティをもつコードストリームの段階的復号化

〔図4〕解像度スケーラビリティをもつコードストリームの段階的復号化

間への非可逆色変換しかありませんでしたが，JPEG2000では式(3)で与えられる可逆色変換や，色変換を行わないオプションも存在しています．

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0.229 & 0.587 & 0.114 \\ -0.16875 & -0.33126 & 0.5 \\ 0.5 & -0.41869 & -0.08131 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \cdots\cdots(2)$$

$$\begin{bmatrix} Y' \\ C_b' \\ C_r' \end{bmatrix} = \begin{bmatrix} \left\lfloor \dfrac{R+2G+B}{4} \right\rfloor \\ R-G \\ B-G \end{bmatrix} \cdots\cdots(3)$$

ただし，$\lfloor x \rfloor$はxを超えない最大の整数とします．

▶タイリング

JPEG2000では，画像を「タイル」と呼ばれる長方形の領域に分割し，それぞれのタイルを独立に符号化するオプションが存在します．この画像をタイルに分割する処理をタイリングと呼びます．

このタイリングによるメリットはおもに2点あり，一つはそれぞれのタイルに対し，量子化係数やエラー耐性オプションなど独立した符号化オプションを与えられること，もう一つは画像をタイルのサイズに領域分割することにより，後述のウェーブレット変換においてバッファリングに使用されるメモリのサイズを小さくすることができることです．

一方，タイリングのデメリットとしては，タイリングを行わない場合と比較して，高圧縮率時にタイル境界での歪みが目立つことがあげられます(**図6**)．

▶ウェーブレット変換(DWT)

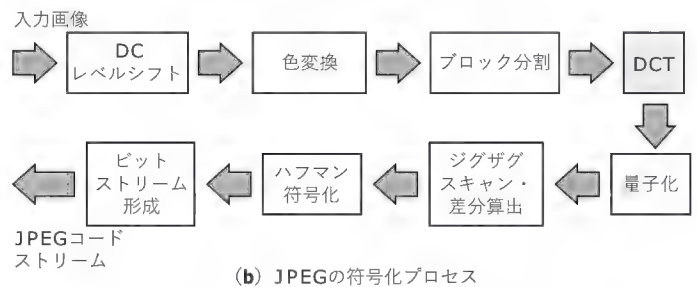
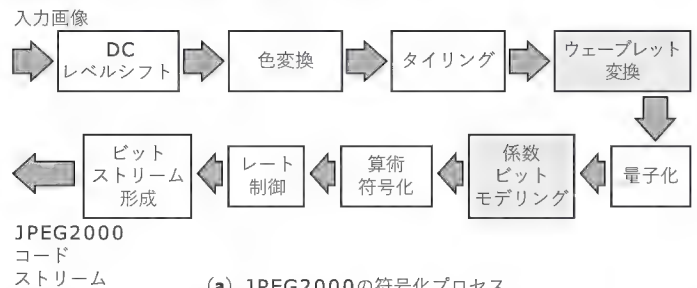
JPEG2000においてもっとも注目されているアルゴリズムの一つに，このウェーブレット変換(DWT：Discrete Wavelet Transform)があります．JPEG2000基本方式では整数型と実数型の2種類のDWTが定義されています．整数型DWTは可逆変換であり，実数型DWTよりも計算量が約半分と少なくなっています．実数型DWTは小数の計算を含み計算量も多いのですが，整数型DWTよりも圧縮効率が高く，同一圧縮率では画質が良くなります．

〔図5〕JPEG2000/JPEGの符号化プロセス

(a) JPEG2000の符号化プロセス

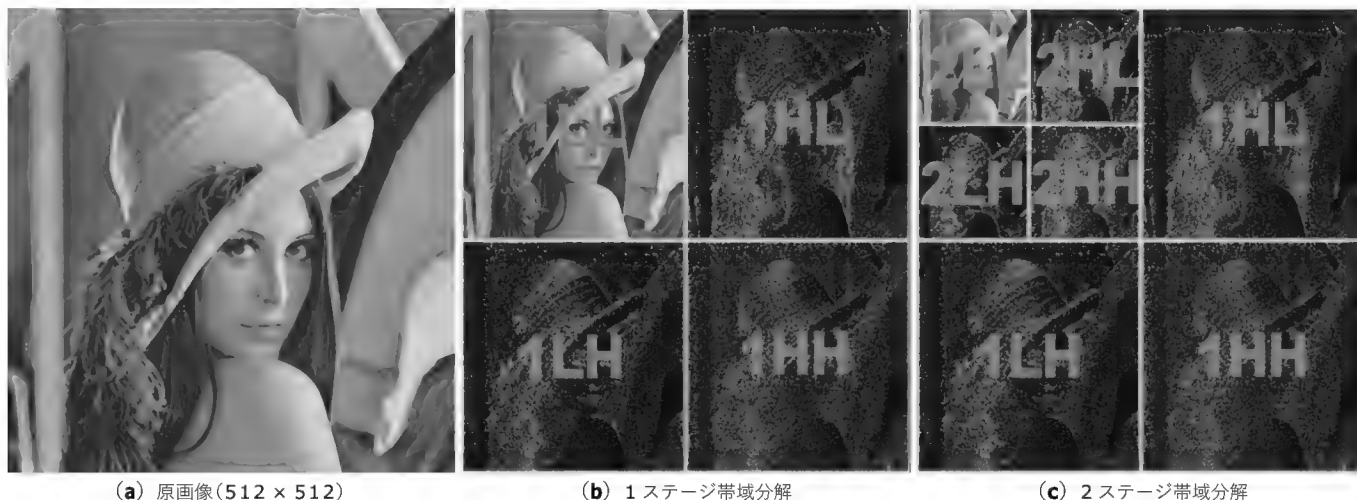
(b) JPEGの符号化プロセス

〔図6〕タイリング処理の有無による歪み発生

(a) 1/120圧縮 タイリングなし　(b) 1/120圧縮 タイリングあり

〔図7〕DWTによる画像のサブバンドへの帯域分割

(a) 原画像(512 × 512)

(b) 1ステージ帯域分解

(c) 2ステージ帯域分解

JPEGでは，RGB色空間からYC_bC_rと呼ばれる輝度成分と色差成分からなる色空間への変換の後，それぞれの成分について，8 × 8画素のブロックに分割されました．そして各ブロックについて2次元DCT(Discrete Cosine Transform，離散コサイン変換)が施されました．このブロック分割から2次元DCTまでの過程が，圧縮率を高くしたときに生じるブロック歪みの原因となっていました．

一方，JPEG2000では，JPEGと同様のRGB色空間からYC_bC_r色空間への変換，または可逆の色変換の後，後述の2次元DWTが施されます．JPEGの場合と異なり，この過程ではブロック歪みを生じません．このため一般的に「JPEG2000はDWTを使用しているからブロック歪みを生じない」と考えられがちなのですが，正しい認識としては，「DWTはブロック分割を必要とせず，JPEG2000はブロック分割を行わないのでブロック歪みを生じない」となります．このことについて，両者の計算式を用いて解説します．

信号長Nの1次元の入力信号$x(n)$，$n = 0, 1, ..., N-1$を想定したとき，式(4)に基本的なDCTの1次元の計算式が，式(5)に整数型DWTの1次元の計算式を示しています．

$$y(k)=\sqrt{\frac{2}{N}}C_k\sum_{n=0}^{N-1}x(n)\cos\frac{(2n+1)k\pi}{2N},\ C_k=\begin{cases}\frac{1}{\sqrt{2}} & (k=0)\\ 1 & (k\neq 0)\end{cases} \quad \cdots\cdots\cdots(4)$$

$$\begin{aligned}y_L(k)&=(-x_{ext}(k-2)+2x_{ext}(k-1)\\&\quad+6x_{ext}(k)+2x_{ext}(k+1)-x_{ext}(k+2))/8 \quad \cdots\cdots\cdots(5)\\ y_H(k)&=(-x_{ext}(k-1)+2x_{ext}(k)-x_{ext}(k+1))/2\end{aligned}$$

ただし，x_{ext}は入力信号xの両端に折り返し処理を施し拡張した信号，y_Lは出力信号の低周波成分，y_Hは高周波成分とします．

式(5)の割り算の部分を除けば，どちらの変換の出力信号とも入力信号と係数値の積の和となっています．ただし，式(5)のDWTの係数はすべて定数であるのに対して，式(4)のDCTの係数は入力信号の長さNに依存しています．係数が可変であると，実装上，高速化しづらい，メモリ消費量が一定でないといった点で不利になるので，JPEGではN=8として，DCTの係数を定数化しています．ここでブロック分割の必要性が生じ，ブロック歪みの原因となります．よって，かりに可変長で，画像と同じ大きさのDCTフィルタリングを施せば，DCTでもブロック歪みは発生しないことになります．

一方，DWTでは入力信号の始めと終わりの両端に折り返し処理を施せば，信号長による計算への影響はないので，実装上でもJPEGのようにブロック分割によって信号長を制限する必要がありません．よって，JPEG2000ではJPEGでよく見られたブロック歪みが発生しないことになります．この特徴は2種類のDWT，整数型，実数型のどちらでも変わりません．

各色成分はDWTによって，サブバンドに帯域分割されます．**図7**はDWTによる帯域分解の一例で，**図7(a)**の原画像に対して2次元DWTを1回施すと**図7(b)**に，さらにそのLLサブバンドに対して2次元DWTを施すと**図7(c)**のように帯域分割されていきます．

▶量子化

JPEG2000ではこの量子化の処理はオプションとなっており，前述のウェーブレット変換で整数型DWTが選択されたときは，量子化は行われません．JPEGでもそうでしたが，この量子化のプロセスは非可逆の処理です．ですからJPEG2000では，量子化を行わず，整数型DWTを用いることによって可逆圧縮を実現しています．参考までに，非可逆圧縮時には式(6)で与えられる処理が施されます．

$$q_b(u,v)=\mathrm{sign}(a_b(u,v))\cdot\left\lfloor\frac{|a_b(u,v)|}{\Delta_b}\right\rfloor \quad \cdots\cdots\cdots(6)$$

$$\Delta_b=2^{R_b e_b}(1+\frac{\mu_b}{2^{11}})$$

ただし，$q_b(u, v)$は量子化後の係数，
sign(x)はxの(正負の)符号，

R_bはサブバンドbのダイナミックレンジ,

e_b, μ_bは量子化パラメータで，ヘッダ部分にその値が保持されます．

▶係数ビットモデリング

各サブバンドの係数は必要に応じて量子化され，EBCOT (Embedded Block Coding with Optimized Truncation)と呼ばれるエントロピー符号化が行われます．EBCOTはコードブロック分割，係数ビットモデリング，算術符号化のプロセスから構成されており，ここでは係数ビットモデリングを中心に説明します．

量子化された各係数は，**図8**が示すようにサブバンドごとにコードブロックに分割されます．**図8**では原画像が512 × 512画素で，コードブロックが64 × 64画素なので，64個のコードブロックに分割されることになります．この分割された各コードブロックに対して以降のエントロピー符号化の処理が行われます．コードブロックの大きさは，各辺が4以上の2のべき乗であり，面積が4096を超えない範囲で自由に決めることができますが，上記の64 × 64または32 × 32が一般的です．

次に，各コードブロック内の量子化された係数値を符号ビット(係数値が負なら1，正か0なら0の値をもつ)と2進数により表現された絶対値のビットプレーンに分離します．コードブロック内の係数を上位ビットプレーンから走査していき，どこかでビット1が含まれるビットプレーン，すなわちそのコードブロックのMSB (Most Significant Bit-plane)を見つけます．そして，そのMSBから下位ビットプレーンの方向に以下の手順でビットプレーンごとに符号化されていきます．

まず，各ビットプレーンにおいて，**図9**に示すように，左上から垂直方向に4係数が走査され，一列ずつ右にずれながら垂直方向に4係数を走査し続けます．最右列の走査が終わったら下の段に移り，一段目で行われたように走査を続けます．そして，走査によって得られた各係数の2値情報は，レート制御を行うときの画質を考慮して，その係数の周囲状態やその係数のMSBの状態によって，次の3種類の符号化パス(サブビットプレーン)に分解，整列されます．

1) Significance Propagation Pass：周囲に有意な係数が一つでも存在するが，その係数自体はまだ有意でない係数に対して適用されるパス

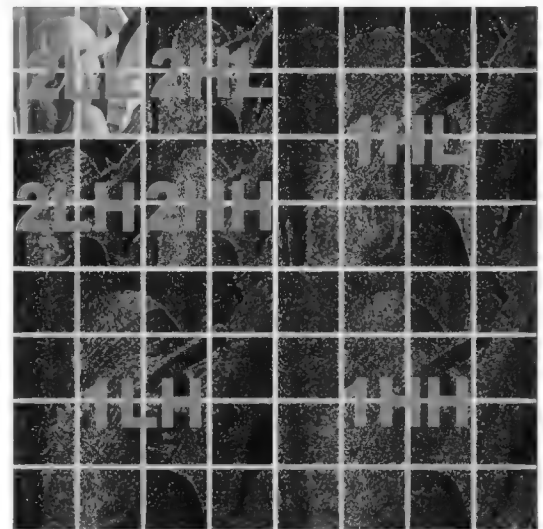
2) Magnitude Refinement Pass：その係数自体がすでに有意である係数に対して適用されるパス

3) Cleanup Pass：上の二つのパスに当てはまらない係数に適用されるパス

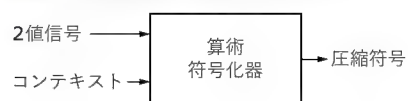
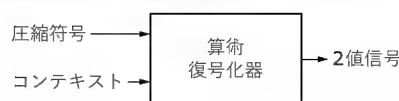
※有意：符号化のプロセスの中での各係数の状態で，「有意である(1)/ない(0)」の2値情報で表されます．符号化の初期段階ではすべての係数は「有意でない(0)」という情報をもちます．そしてMSBから各係数の情報を1ビットずつ符号化し，係数内で1が初めて符号化されたとき，つまりその係数の最重要ビットが符号化されたときにその係数は「有意である(1)」という状態に変化し，以後その係数は有意であり続けます．また，最重要ビットが符号化された場合，続いてその係数が正であるか負であるかの情報が符号化されます．

以上の三つの符号化パスに分類された各係数の当該ビットプレーンにおける2値情報は，**図10**に示すように，その係数が置かれている状態を示すコンテキストを入力として算術符号化器に送られ，圧縮された符号を出力として得ます．

ここでコンテキストに関して簡単に説明します．コンテキス

〔図8〕64 × 64コードブロック分割

〔図9〕コードブロックの係数走査パターンの例(横16の場合)

0	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60
1	5	9	13	17	21	25	29	33	37	41	45	49	53	57	61
2	6	10	14	18	22	26	30	34	38	42	46	50	54	58	62
3	7	11	15	19	23	27	31	35	39	43	47	51	55	59	63
64	68	72	76	80	84	88	92	96	100	104	108	112	116	120	124
65	69	73	77	81	85	89	93	97	101	105	109	113	117	121	125
66	70	74	78	82	86	90	94	98	102	106	110	114	118	122	126
67	71	75	79	83	87	91	95	99	103	107	111	115	119	123	127
128															

〔図10〕算術符号化器の入出力モデル

〔図11〕算術復号化器の入出力モデル

〔図12〕注目係数と隣接係数

D_0	V_0	D_1
H_0	X	H_1
D_2	V_1	D_3

トとは，算術符号化器内部において2値情報の確率推定状態を区別するための値で，係数ビットモデリングのプロセスから係数とその周囲の状態をもとにコンテキストラベルとして与えられます．**図11**に算術復号化器のモデルを示しますが，この場合もコンテキストラベルは係数ビットモデリングのプロセスから与えられます．つまり，符号化時と復号化時とで同一のコンテキストラベルの算出方法をとることもできるのです．

続いて，コンテキストラベルの算出方法について説明します．JPEG2000の係数ビットモデリングでは19のコンテキストラベルが存在しますが，ここでは便宜的にそのラベルが0から18までの整数であると仮定します．**図12**に示すように，Xの周囲に存在する八つの係数（H，V，D）の有意性と符号が当該係数Xのコンテキストラベルを算出するのに使われます．ここから，三つの符号化パスについてそれぞれ説明します．

1）Propagation Pass

本パスにおいては，周囲八つの係数の有意性をもとにして，**表1**に基づいてコンテキストラベルを算出します．本パスの定義から，周囲8係数のうち最低1係数は有意であるため，必然的に1から8までの8通りのコンテキストラベルのうち，どれかが算出されます．たとえば，現在のコードブロックがLLバンドにあって，現在の係数の左上（D_0），上（V_0），右上（D_1）の三つの係数だけが有意であり，現係数が有意でない場合には，この係数ビットは本パスにおいて，コンテキストラベル3とともに符号化されます．

また，前にも少し説明しましたが，ある係数内で1が初めて符号化されたとき，すなわちその係数の最重要ビットが符号化されたときには，続いてその係数の正負の符号情報も符号化されます．その際のコンテキストラベルは**表2**に基づいてその係数の垂直・水平成分の関与を求めます．次にその垂直・水平成分の関与の値から**表3**に基づいてコンテキストラベルを算出します．XORビットの値が1のときには，算術符号化器に送るその係数の正負情報（正：0，負：1）の排他的論理和（正：1，負：0）を使用します．たとえば，現在の係数の左（H_0）が有意で正，上（V_0）が有意で負，残りの二つ（H_1，V_1）が有意でない場合は，コンテキストラベルが11，XORビットは0となります．

2）Magnitude Refinement Pass

本パスにおいては，この係数ビットが最重要ビットのすぐ下のビットであるかどうかと，周囲に有意な係数があるかどうかの情報をもとに，**表4**に基づいてコンテキストラベルを算出します．結果的には，その係数が有意となった次のビットプレーンでは周囲の係数によって14か15のコンテキストラベルが与えられますが，それ以降は16が与えられることになります．

3）Cleanup Pass

本パスにおいてはSignificance Propagation Passと同様の符号化のほかに，ランレングス符号化と呼ばれるもう1通りの符号化方法が存在し，その組み合わせで符号化していきます．

〔表1〕Significance Propagation Pass用コンテキストラベル

LL，LHサブバンド			HLサブバンド			HHサブバンド		コンテキストラベル
ΣH_i	ΣV_i	ΣD_i	ΣH_i	ΣV_i	ΣD_i	$\Sigma(H_i+V_i)$	ΣD_i	
2	x	x	x	2	x	x	≧3	8
1	≧1	x	≧1	1	x	≧1	2	7
1	0	≧1	0	1	≧1	0	2	6
1	0	0	0	1	0	≧2	1	5
0	2	x	2	0	x	1	1	4
0	1	x	1	0	x	0	1	3
0	0	≧2	0	0	≧2	≧2	0	2
0	0	1	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0

注）表中のxはこの値は関与しないという意味で，ΣH_i，ΣV_i，ΣD_iはそれぞれ水平，垂直，斜め方向の係数のうちの有意係数の数を示すものとする．また，コンテキストラベルは実装によって異なる値をとることができる

〔表2〕符号コンテキストへの水平/垂直関与

$V_0(H_0)$	$V_1(H_1)$	垂直（水平）関与
有意，正	有意，正	1
有意，負	有意，正	0
非有意	有意，正	1
有意，正	有意，負	0
有意，負	有意，負	−1
非有意	有意，負	−1
有意，正	非有意	1
有意，負	非有意	−1
非有意	非有意	0

〔表3〕水平・垂直関与と符号コンテキストの対応

水平関与	垂直関与	コンテキストラベル	XORビット
1	1	13	0
1	0	12	0
1	−1	11	0
0	1	10	0
0	0	9	0
0	−1	10	1
−1	1	11	1
−1	0	12	1
−1	−1	13	1

JPEG2000のコードブロックの符号化においては，その符号化パスの定義から本パスにおいて2値情報0を連続して符号化するケースが多くなっており，そういったケースでの圧縮効率を高めるために，ランレングス符号化が使われています．

ランレングス符号化は，垂直方向に走査された4係数がすべて本パスに存在し，またその4係数のコンテキストラベルがすべて0であるときに使用されます．そうでないときには，上記のSignificance Propagation Passと同様の符号化方法がとられます．

具体的には，ランレングス符号化は，**表5**に基づいてコンテキストラベルと2値信号が算術符号化器に送られます．たとえば，ある4係数がこのランレングス符号化の条件を満たし，三つ目の係数が1を符号化する場合には，まず信号1とコンテキストラベル17(ランレングス)を算術符号化器に送り，そして信号1とコンテキストラベル18(ユニフォーム)，信号0とコンテキストラベル18を送ります．三つ目の係数が有意となるのは自明なので，続いてその係数の符号を**表2**，**表3**を用いて符号化します．四つ目の係数については他のランレングス符号化しないCleanup Pass上の係数と同様，**表1**に基づいて符号化します．

*　　　　　　*

以上が係数ビットモデリングのアルゴリズムの説明になります．複雑でわかりにくい概念なので，最後におさらいとして，1×4の縦長で係数が{1，5，1，0}のコードブロックがLLサブバンドに存在するものとして，どのようなシンボルとコンテキストのペアが算術符号化器に送られるかについて考えてみましょう．ただし，符号化するうえで，このコードブロックのMSBは下から3ビットプレーン目にあるとします．**表6**がこのコードブロックに対する係数ビットモデリングを図示したものとなります．

▶算術符号化

JPEG2000では，2値算術符号化器であるMQコーダが使用されます．前述した係数ビットモデリングの項でもふれましたが，この算術符号化器はコンテキストラベルと2値情報を入力として，各コンテキストにおいての推定確率とMPS(Most Probable Symbol)値を学習により更新しながら，圧縮された符号を出力とします(**図10**)．

▶レート制御

JPEG2000では量子化によるレート制御のほかに，ポスト量子化とも呼ばれる，符号化パスごとに生成された符号を切り捨てることによるレート制御が存在します．このポスト量子化によるレート制御の段階において，どのコードブロックのどの符号化パスを優先的に残すかの手法が，最終的な画質に大きく影響します．

▶ビットストリーム形成

このビットストリーム形成では，ヘッダなどの情報の付加やこれまでの処理で生成されたビットストリームの並べ替えを行います．JPEG2000ではこのビットストリームの並べ方(プログレッションオーダ)について，以下の5種類が定義されています．

1）LRCP
2）RLCP
3）RPCL
4）PCRL
5）CPRL

ここでは代表的な二つのプログレッションオーダであるLRCPとRLCPについて，簡単に説明します．LRCPのLはレイヤを表し，画質に基づいて設定されます．LRCPによって符号化されたコードストリームを段階的に復号化していくと，**図3**に示したように，画質が徐々に良くなるように復号化されます．

一方，RLCPのRは空間的解像度のレベルを表します．RLCP

〔表4〕Magnitude Refinement Pass用コンテキストラベル

$\Sigma H_i+\Sigma V_i+\Sigma D_i$	最重要ビットの直後のビットか？	コンテキストラベル
x	×	16
≧1	○	15
0	○	14

〔表5〕ランレングス符号化用コンテキストラベルとシンボル

状　況		符号化シンボル	コンテキストラベル
連続した4係数のビットデータがすべて0		0	17(ランレングス)
連続した4係数で1が一つでも符号化される		1	17(ランレングス)
	そのうち初めの1は一つ目の係数	00	18(ユニフォーム)
	そのうち初めの1は二つ目の係数	01	18(ユニフォーム)
	そのうち初めの1は三つ目の係数	10	18(ユニフォーム)
	そのうち初めの1は四つ目の係数	11	18(ユニフォーム)

〔表6〕係数ビットモデリングにより算出されるコンテキストとシンボルの例

ビットプレーン	符号化パス	コンテキストラベル	シンボル	コメント
0	CLパス	17	1	0でないビットをもつ係数が存在
		18	0	初めの0でないビットをもつ係数は二つ目
			1	
		9	0	二つ目の係数は正
		3	0	三つ目の係数のビットは0
		0	0	四つ目の係数のビットは0
1	SPパス	3	0	一つ目の係数のビットは0
		3	0	三つ目の係数のビットは0
	RFパス	14	0	二つ目の係数のビットは0
	CLパス	0	0	四つ目の係数のビットは0
2	SPパス	3	1	一つ目の係数のビットは1
		10	0	一つ目の係数は正
		3	1	三つ目の係数のビットは1
		10	0	三つ目の係数は正
		3	0	四つ目の係数のビットは0
	RFパス	16	1	二つ目の係数のビットは1

※CLパス(Cleanup Pass)，SPパス(Significance Pass)，RFパス(Magnitude Refinement Pass)

によって符号化されたコードストリームを段階的に復号化していくと，**図4**に示したように，解像度が徐々に増していくように復号化されます．

DSPへの実装

ここまで，JPEG2000の特徴やその符号化アルゴリズムについて説明してきました．ここからはいよいよJPEG2000デコーダの実装方法について説明しますが，その前にDSPの特徴についておさらいしておきましょう．

● DSPのおもな特徴

ここではとくにJPEG2000に代表されるマルチメディア関係のソフトウェアを実装するうえで気をつけておかなければならないDSPの特徴について説明します．

▶積和演算を得意とする

DSPの最大の特徴として，積和演算(MAC：Multiply Accumulate)を1クロックで実行できるということがあげられます．DSPは1クロックで積和演算が可能な積和演算器と，1クロックで同時にデータを読み込める高速内部メモリを有することで，積和演算を1クロックで実行できます．

普通の足し算や論理演算などにも1クロックかかるので，アルゴリズム内の計算を可能なかぎりMACを用いた計算に置き換えることで，DSP上でのアルゴリズムの高速化が期待できます．さらに最近では，複数個の積和演算器をもったDSPも登場しているので，DSPでのMAC演算の効率はいっそう高まっています．

▶分岐命令を苦手とする

積和演算をはじめとする高速演算が売り物のDSPですが，分岐命令に対しては，実行に数クロックかかってしまいます．積和演算が1クロックで実行できていたことを考慮すると，DSPは相対的に分岐命令が苦手だといえます．したがって，アルゴリズム内の分岐命令を極力減らすことが，そのアルゴリズムをDSPへ実装する際には不可欠な作業となります．

▶限られた内部メモリ

高速アクセス可能な内部メモリのおかげで，DSPはMACなどの高速演算が行えることは説明しました．また，最近のDSPはさらに大容量の内部メモリを搭載するようになってきました．しかし，画像や音声などを処理するマルチメディアアプリケーションは，その圧縮やフィルタリングの処理に使われるテーブルやパラメータなどを保存しておくために，多大なメモリを使用します．こういったテーブルやパラメータは頻繁に使われることが多いので，本来ならば高速な内部メモリに格納しておくのが望ましいのですが，画像や音声の処理では内部メモリが足りなくなることがしばしばあります．

ですから，アルゴリズムで使われるテーブルやパラメータの大きさをできるかぎり小さくし，使用するメモリ量を最小限におさえることが，DSPへ実装するうえでの重要なポイントとなります．

● JPEG2000デコーダの実装

一般的に，JPEG2000はJPEGと比較して圧縮で約5倍，伸張で約3倍の処理能力が必要といわれています．さらに，DSPへの実装という観点からすると，JPEGのDCTはDSPが得意としていた積和演算で構成されていましたが，JPEG2000の整数型ウェーブレットは足し算とビットシフトで構成されており，また，係数ビットモデリングから算術符号化/復号化にかけてのアルゴリズムは，DSPがまさに苦手とする分岐命令の連続といっても過言ではありません．ですから，一般的にJPEG2000はDSPにはあまり向いていない符号化アルゴリズムであると考えられています．しかしここでは，JPEG2000デコーダの中で，本来DSPがあまり得意としない，前述した代表的な二つのアルゴリズムについて，いま挙げたDSPの特徴をふまえつつ，どのようにすればDSPにとって最適化されたアルゴリズムとして実装できるかについて説明します．

▶逆ウェーブレット変換(IDWT)

まずは逆ウェーブレット変換のDSPへの実装方法，ここではとくに，可逆圧縮を可能にした整数型IDWTの実装方法について紹介します．

整数型DWTの変換式は式(5)に示しましたが，実際には，計算量を減らすために，リフティング構成による整数型DWTが実行されます．信号長Nの1次元の入力信号$x(n)$，$n=0, 1, \ldots, N-1$を想定し，$x(n)$の両端に折り返し処理を施して拡張した信号を$X_{ext}(n)$とすると，リフティング構成による整数型DWTの変換式は式(7)のようになります．また，この逆変換に相当する，リフティング構成による整数型IDWTの変換式は，式(8)に見られるように演算的には非常に似通った構成になっています．ここでは，式(8)にある整数型IDWTのうち，2段目の演算のDSPへの実装について考えてみます．

$$Y(2n+1)=X_{ext}(2n+1)-\left\lfloor\frac{X_{ext}(2n)+X_{ext}(2n+2)}{2}\right\rfloor$$
$$Y(2n)=X_{ext}(2n)+\left\lfloor\frac{Y(2n-1)+Y(2n+1)+2}{4}\right\rfloor \quad \cdots\cdots(7)$$

$$X(2n)=Y_{ext}(2n)-\left\lfloor\frac{Y_{ext}(2n-1)+Y_{ext}(2n+1)+2}{4}\right\rfloor$$
$$X(2n+1)=Y_{ext}(2n+1)+\left\lfloor\frac{X(2n)+X(2n+2)}{2}\right\rfloor \quad \cdots\cdots(8)$$

最初に，この演算がDSPではどのように扱われるかについて考えてみます．

1) まず，$X(2n)$の値をメモリからアキュムレータにロードする
2) アキュムレータに$X(2n+2)$の値を加える
3) アキュムレータの値を右側に1ビットシフトする
4) アキュムレータの値に$Y_{ext}(2n+1)$の値を加える
5) アキュムレータの値を$X(2n+1)$としてメモリにストアする

ここでは，3)の過程によって，2で割ってfloorの値をとるという演算が置き換えられました．DSPは積和演算が得意ですが，

割り算は得意ではありません．よって，とくにこの場合のように分母が2のべき乗である場合には，ビットシフトによって割り算を置き換えるというのが最適化の一つのポイントとなります．

この一連の演算をテキサス・インスツルメンツ社（以下，TI）の低消費電力の固定小数点DSP，TMS320C55x（以下，C55x）用のアセンブラで書くと**リスト1**のようになり，この一つの係数に対する演算は5クロックかかることがわかります．

では，これ以上の最適化はできないのでしょうか．さきほど述べたDSPの特徴をもう1度思い出してみましょう．DSPは積和演算が得意なプロセッサでした．つまり，積和演算を多用するほどその計算効率は高まります．この特徴を利用して，ここでは上記のDWTの演算のうち，1)～4)を積和演算に置き換える形での最適化を考えてみましょう．

1) $Y_{ext}(2n+1)$の値をメモリからアキュムレータの上位16ビットにロードする
2) アキュムレータ，$X(2n)$の値，0x4000で積和演算を行う
3) アキュムレータ，$X(2n+2)$の値，0x4000で積和演算を行う
4) アキュムレータの上位16ビットの値を$X(2n+1)$として，メモリにストアする

ここでは，2)，3)の過程において，足し算＋ビットシフトの演算が積和演算に置き換えられました．メモリの値を右側に1ビットシフトすることと，メモリの値に0x4000を掛けてアキュムレータの上位16ビットを抽出することが同値であるということが，ここでのポイントです．

同様にこの演算をC55x用のアセンブラで表記すると**リスト2**のようになり，この計算にかかる演算が4クロックに減ったことがわかります．

また，最近のDSPでは，1クロックで複数の積和演算を実行することが可能となっており，C55xでも二つの積和演算を1クロックで実行することが可能です．この性能をうまく利用すれば，**リスト2**に示した4クロックかかる演算のさらなる高速化も可能です．

これらの一連の高速化は，積和演算を用いた計算に極力置き換たことに起因しています．つまり，DSPに最適化されたソフトウェアを実装するためのポイントの一つは，積和演算を効率よく用いることだということを覚えておいてください．

▶係数ビットモデリング

次に，係数ビットモデリングの処理の実装方法についてですが，ここではSignificance Propagation Pass，およびCleanup Passにおいて，周囲八つの係数の有意性からコンテキストラベルを算出する部分について紹介します．

周囲8係数の有意性からコンテキストラベルを算出する方法については前述しましたが，理論的には，垂直方向，水平方向，斜め方向でそれぞれ有意な係数の数の和を求め，その値を**表1**に当てはめてコンテキストラベルを算出します．しかし，もしこの処理を分岐命令のみを使って，理論どおりにコンテキストラベルを算出しようとすると，まず周囲8係数に対してそれぞれ1回ずつ，サブバンドの場合分けに対して1回，テーブル内での場合分けに対して最低数回と，一つのコンテキストラベルを算出するのに十数回の分岐命令が必要となります．この処理のままでは，実装するのがDSPにしても他のプロセッサにしても，処理が重すぎて現実的ではありません．そこで，まずはこの分岐命令を減らす方向での実装方法を考えてみます．

〔リスト1〕積和演算を用いないDWTの実装例

```
初期設定：*AR3=Yext[i+1]，T0=2，*AR2=X[i]
1) AC0 = *(AR2+T0)
2) AC0 = AC0 + *AR2-
3) AC0 = AC0 << #-1
4) AR1 = *AR3 - AC0
5) *AR2+ = AR1
```

〔リスト2〕積和演算を用いたDWTの実装例

```
初期設定：*AR3=Yext[i+1]，T0=2，*AR2=X[i]，*AR1=0x4000
1) hi(AC0) = *(AR3+T0)
2) AC0 = AC0 + (*AR1 * *(AR2+T0))
3) AC0 = AC0 + (*AR1 * *AR2-)
4) *AR2+ = hi(AC0)
```

このような複雑な操作を簡略する一般的な方法には，ルックアップテーブル（LUT）と呼ばれるものがあります．LUTとは基本的に，関数の演算結果を入力変数ごとにあらかじめ計算して記憶しておく方法です．実際の演算時には，LUTの入力変数にあたる位置のデータを参照することにより，複雑な演算を単純な参照の処理に置き換えることができます．このLUTを用いたコンテキストラベルの算出を考えてみましょう．

ここではLUTに対する入力として，周囲8係数の有意性とサブバンドの種類（LL，LHなど）が考えられます．各係数の有意性はある（1），ない（0）のように1ビットの情報で表され，それが8係数あるので，8ビットです．サブバンドの種類はLL，HL，LH，HHの4種類があるので2ビットです．合計して，各係数に10ビットの情報が必要となります．

LUTに対する入力情報が10ビットあるので，LUTは2^{10}＝1024通りの出力をもたなければなりません．すなわちこのLUTには1024ワードのメモリが用意され，1024通りの入力に対する出力をあらかじめ計算して保持しておきます．

一方で，コードブロックと同じ大きさのメモリをこのビットデータ用にバッファとして確保し，**図13**に示すように，その下位2ビットにサブバンドの種類（LL：0，HL：1，LH：2，HH：3）を，それの上位8ビットに各係数の有意性を保持しておくことにします．具体的には，初期化の処理において，サブバンドの種類を示す2ビットのデータをバッファに記憶させます．その後は係数が非有意から有意になるたびに，バッファ内にあるその係数の周囲8係数の有意ビット情報を更新します．そして，コンテキストラベルの値が必要なときには，LUTの中で当該係数の10ビットデータが指し示す位置に保持された値を参照します．

具体例をあげながら説明します．たとえばこのコードブロッ

クが属しているサブバンドの種類がLHで，この係数の周囲8係数のうち，上と左下と右下の係数がすでに有意であるとします．すると，**図14**が示すように，この係数のこの時点でのビットデータは278となります．よって，LUTの278番目のデータを参照して，この場合は3をコンテキストラベルとして使用します．

このようなLUTを用いた実装により，比較的複雑だったコンテキストラベルを計算する処理を簡略化できました．DSPの苦手とする分岐命令もLUTによって完全に置き換えられ，かなりの高速化が期待できます．ここでは係数ビットモデリングのコンテキスト計算の一つを例としてあげましたが，他の計算でも同様の処理によって高速化が期待できます．ここでのポイントは，分岐命令などを用いた複雑な計算においては，LUTを用いることによって，高速化が望める場合があるということです．

〔図13〕LUTの入力に使われるビットデータの配置例

← MSB									LSB →
9	8	7	6	5	4	3	2	1	0
左上	上	右上	左	右	左下	下	右下	サブバンド	
周囲の係数の有意性(各1ビット)								サブバンドの種類(2ビット)	

〔図14〕LUTの入力に使われるビットデータの例
(LHサブバンドで上，左下，右下の係数が有意な場合)

9	8	7	6	5	4	3	2	1	0
左上	上	右上	左	右	左下	下	右下	サブバンド	
0	1	0	0	0	1	0	1	1	0

LUTへの入力データは$(0100010110)_2$=278となる

〔リスト3〕係数ビットモデリング用省メモリLUTの実装例

```
const short LUTupdateHV[9] = {
  (1<<8) | (3<<4) | (5),
  (2<<8) | (3<<4) | (6),
  (2<<8) | (3<<4) | (6),
  (4<<8) | (4<<4) | (7),
  (5<<8) | (4<<4) | (7),
  (5<<8) | (7<<4) | (8),
  (7<<8) | (7<<4) | (8),
  (7<<8) | (7<<4) | (8),
  (8<<8) | (8<<4) | (8) };
```

(a) 水平垂直方向コンテキストラベル更新テーブル

```
const short LUTupdateD[9] = {
  (3<<8) | (1<<4) | (1),
  (4<<8) | (2<<4) | (2),
  (5<<8) | (2<<4) | (2),
  (6<<8) | (3<<4) | (3),
  (7<<8) | (4<<4) | (4),
  (7<<8) | (6<<4) | (6),
  (8<<8) | (6<<4) | (6),
  (8<<8) | (7<<4) | (7),
  (8<<8) | (8<<4) | (8) };
```

(b) 斜め方向コンテキストラベル更新テーブル

```
const short LUT HDshift[4] = {
/* LL band */   0,
/* HL band */   4,
/* LH band */   0,
/* HH band */   8};
```

(c) 水平斜め方向テーブル内シフト値テーブル

```
const short LUT Vshift[4] = {
/* LL band */   4,
/* HL band */   0,
/* LH band */   4,
/* HH band */   8}
```

(d) 垂直方向テーブル内シフト値テーブル

しかし，このLUTにも問題がないわけではありません．とくにDSPにとって問題となりやすいのはLUT自体のサイズです．高速化というLUTの役割を考えた場合，LUTは高速アクセスが可能なDSPの内部メモリに格納されている必要があります．しかし前にも説明しましたが，DSPの内部メモリは限られており，とくにJPEG2000をはじめとするマルチメディアアプリケーションを実装する際には不足しがちです．こうした実情を考慮したとき，前出の例のようにLUTに1024ワードものメモリを使用するのは得策とはいえません．そこでここでは，LUTによる高速化のメリットをできるかぎり損なうことなく，かつLUTに使われるメモリサイズを小さくする方向での最適化について考えてみましょう．

そもそも，すべての係数に与えられるコンテキストラベルの初期値は0です．それから，周囲の係数が有意になっていくことによって，コンテキストラベルは更新されていくと考えることもできます．そこで今回は，コンテキストラベルを算出するLUTの代わりに，周囲の係数が有意となったときにコンテキストラベルを更新していくLUTの実装を行います．

LUTのサイズをできるかぎり小さくするための実装方法の一例が，**リスト3**に示されている4種類のLUTを用いたものです．前出の実装例では周囲の有意性やサブバンドの種類のデータを保持していたバッファに対して，今回はまずコンテキストラベルの初期値である0をすべての係数に対して記憶させます．

コンテキストラベルの算出には，このバッファ内の値を参照し，周囲の係数が有意となったときには，**リスト3**のテーブルを用いて，以下の手順でバッファ内のコンテキストラベルを更新します．まず，**リスト3(c)**，**(d)**から，現コードブロックが属するサブバンドの種類を入力変数として，**リスト3(a)**，**(b)**のテーブルのどの部分を更新されたコンテキストラベルとして使用するかを，テーブルの値に対するシフト値として決定します．次に，**リスト3(a)**，**(b)**のテーブルに元のコンテキストラベルとシフト値を入力変数として，新しいコンテキストラベルに更新します．この処理を，係数が有意となるたびに，その周囲8係数に対して施します．

この一連の処理について，具体例をあげて説明します．たとえば現在のコードブロックが属しているサブバンドの種類がHLで，有意となろうとしている係数Xの周囲8係数のコンテキストラベルが**図15(a)**のようになっているものとします．まず**リスト3(c)**，**(d)**から，水平と斜め方向成分のシフト値4と垂直方向成分のシフト値0を得ます．次に，垂直水平斜めの各コンテキストラベルに対して，**リスト3(a)**，**(b)**のテーブルに元のコンテキストラベルを入力変数として，テーブル値を参照します．最

〔図15〕省メモリLUTを用いたコンテキストラベル更新処理の例

7	6	2
3	X	3
1	0	1

(a) 更新前のコンテキストラベル

7	8	2
4	X	4
2	5	2

(b) 更新後のコンテキストラベル

後に，参照されたテーブル値をシフト値分だけ右側にシフトし，下位4ビットを抽出することによって，更新されたコンテキストを得ることができます．この一連の処理は，**リスト4**のように書かれ，結果**リスト3(b)**のように周囲8係数のコンテキストラベルが更新されることになります．

このように省メモリのLUTを用いることによって，LUTのサイズは1024ワードから計26ワードへと，約1/40の大きさに縮小されました．今回の実装のように，高速化をめざすだけでなく，極力メモリスペースを使わない実装方法を考案することも，DSPにソフトウェアを実装するうえでの重要なポイントです．

おわりに

画像や音声の圧縮・伸張を始めとした，最近のマルチメディア信号処理は多様化の一途をたどっており，またコストの低減や開発期間の短縮といった要求も強いことから，こうした状況に対応できるDSPの必要性が高まっています．

その一方で，こうしたマルチメディア信号処理は複雑化しており，DSPがあまり得意としない処理を多用するものが多くなってきました．

こうした状況をふまえ，本稿ではDSPを用いた最近の画像処理技術の一例として，JPEG2000デコーダのDSPへの実装方法について解説しました．一見してDSPが得意としない処理やアルゴリズムでも，DSPの特徴をふまえた工夫を凝らすことによって，DSPに最適化された実装が可能であり，DSPへの実装というソリューションが十分に有力なものであるということをわかっていただけたことと思います．

〔リスト4〕省メモリLUTを用いたコンテキスト値更新の実装例

```
/* code block width = コードブロックの横幅 */
/* *ctxp は初期状態で左上のコンテキスト値を指しているものとします. */
shiftHD = LUT_HDshift[subband_type]; /* 水平斜め方向のシフト値 */
shiftV  = LUT_Vshift[subband_type];  /* 垂直方向のシフト値 */

/* 左上のコンテキスト値の更新 */
*ctxp = (LUTupdateD[*ctxp] >> shiftHD) & 0x0f;
ctxp++;

/* 上のコンテキスト値の更新 */
*ctxp = (LUTupdateHV[*ctxp] >> shiftV) & 0x0f;
ctxp++;

/* 右上のコンテキスト値の更新 */
*ctxp = (LUTupdateD[*ctxp] >> shiftHD) & 0x0f;
ctxp += code_block_width;

/* 右のコンテキスト値の更新 */
*ctxp = (LUTupdateHV[*ctxp] >> shiftHD) & 0x0f;
ctxp -= 2;

/* 左のコンテキスト値の更新 */
*ctxp = (LUTupdateHV[*ctxp] >> shiftHD) & 0x0f;
ctxp += code_block_width;

/* 左下のコンテキスト値の更新 */
*ctxp = (LUTupdateD[*ctxp] >> shiftHD) & 0x0f;
ctxp++;

/* 下のコンテキスト値の更新 */
*ctxp = (LUTupdateHV[*ctxp] >> shiftV) & 0x0f;
ctxp++;

/* 右下のコンテキスト値の更新 */
*ctxp = (LUTupdateD[*ctxp] >> shiftHD) & 0x0f;
```

参考文献

1) ISO/IEC FDIS15444-1, "Information Technology - JPEG2000 Image Coding System - Part-1: Core Coding System", ISO/IEC JTC1/SC29/WG1 Jan. 2001
2) 貴家仁志，「JPEG2000を中心とした画像圧縮技術の新しい流れ」，『Interface』，2002年1月号，pp.46-58
3) 貴家仁志/渡邊修，「JPEG2000符号化アルゴリズムの要素技術」，『Interface』，2002年1月号，pp.59-71
4) 福原隆浩，「JPEG2000/Motion-JPEG2000の技術概要と応用 前編」，『Interface』，2002年11月号，pp.131-142

しま・まさと　日本テキサス・インスツルメンツ(株)

第17回

Perlの統合開発環境—「Open Perl IDE」と「Perlを始めよう！」

水野貴明

Perlは世界中で広く使われているプログラミング言語である．CGI(Common Gateway Interface)にもよく利用されているし，定型作業の自動化，ちょっとしたテキスト処理などの際にも，非常に便利だ．

しかし，Perlは基本的にコマンドラインで利用することを前提としたツールである．あたりまえといえばあたりまえだが，GUI全盛のこの時代，Perlを用いた開発も，GUIの機能をもっと利用した開発環境で行いたい気もする．そこで今回は，Windowsで利用できるPerlの統合開発環境(IDE)を二つ，紹介したいと思う．

Open Perl IDE

DATA

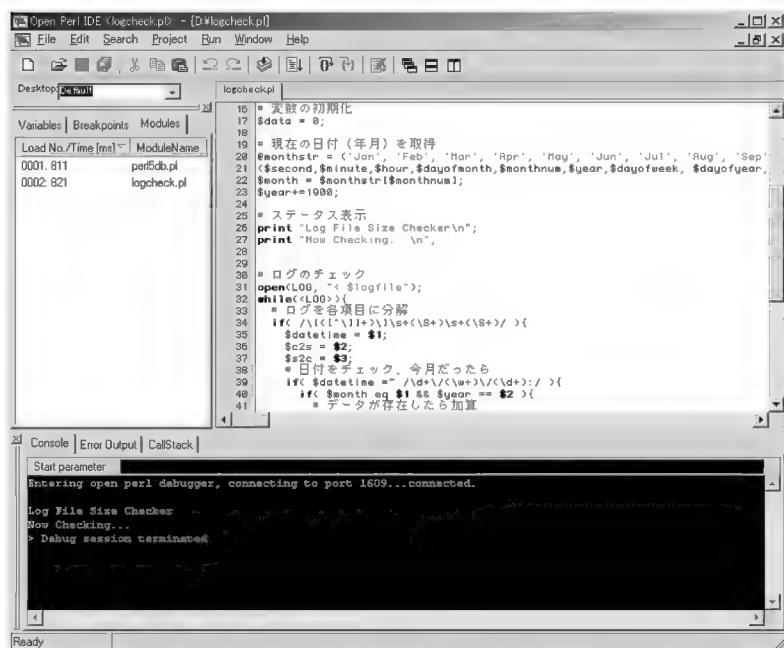
名称：Open Perl IDE
作者：Jürgen Güntherodt
Webサイト：http://open-perl-ide.sourceforge.net/
現在のバージョン：Ver.1.0(2002/5/30)
ダウンロードサイズ：1073Kバイト(zip圧縮ファイル)
OS：Windows 95/98/NT4/2000/XP

〔図1〕Open Perl IDEの作業画面

まずはじめに紹介するのは，Open Perl IDEである．これは，その名のとおり，オープンソースで開発が進められている，ドイツ生まれの強力な開発環境である．

インストールと環境設定

Open Perl IDEは，SourceForge.net[注1]で公開されている．ダウンロードできる圧縮ファイルには，Open Perl IDE本体の実行ファイル，デバッグ用の「perl5db.pl」，ヘルプファイル，Open Perl IDEのソースコードなどが入っている．とくにインストーラなどはついておらず，展開した実行ファイルを実行すれば，IDEが起動する．ただし，Perlそのものがインストールされるわけではないので，別途インストールしておく必要がある．今回の検証に用いたのは，ActiveState社[注2]が公開しているActivePerl(v5.6.0 Binary Build 623)である．

Open Perl IDEの画面は**図1**のようになっている．このIDEは，さまざまな機能をもつ複数のウィンドウで構成されており，ウィンドウの役割は**表1**のとおりである．それぞれのウィンドウはフローティングウィンドウとして利用できるほか，メインウィンドウであ

注1：オープンソース開発者向けのCVS(Concurrent Versions System)リポジトリ，Webサイトやメーリングリスト，バグトラッキングシステムなどを提供するサイト．日本版のSourceForge.jpもある．

注2：http://www.activestate.com/

るソースコードエディタの上にドラッグすることで，ソースコードエディタとドッキングすることも可能である(フローティング&ドッキング機能[注3])．各ウィンドウのソースコードエディタ上での配置も自由に変更が可能で，すべてのフローティングウィンドウを一つにまとめてタブで切り替えることもでき，自由に自分の使いやすい開発環境を構築できるようになっている(**図2**)．

さて，Open Perl IDEを使うにあたって，まずやらなければならないのはフォントの指定である．Open Perl IDEではエディタで利用するフォントとして，標準でCourier Newが選択されているが，これでは日本語の表示ができない．そこで表示用のフォントを変更して，日本語が表示できるようにするのだ(もちろん，日本語をまったく利用しないのであれば，変更の必要はない)．「Edit」メニューの「Preferences...」を選択すると，設定ダイアログ(**図3**)が表示される．このウィンドウでは，エディタのフォント設定以外にも，各種ウィンドウでのフォントやPerlのキーワードのハイライト，タブの長さなどの設定が可能になっている．

また，設定ウィンドウではPerlの実行ファイル(`perl.exe`)のパスを指定する項目がある．Open Perl IDEを初めて起動した状態ではここには何も入っていないが，とくにPerlの場所を指定しなくても，Windowsの設定でPerlにパスが通っていれば，IDE上からのプログラムの実行が可能である．しかも，一度プログラムの実行を行うと，きちんとパスが設定された状態になる．もし，Open Perl IDEがPerlの実行ファイルを見つけることができず，プログラムの実行ができなかった場合には，自分で正しいパスを指定する必要がある．

〔表1〕Open Perl IDEで用意されているウィンドウ

Console	プログラムの出力結果を表示するウィンドウ
Error Output	エラーメッセージを表示する
Breakpoints	設定されたブレークポイントを表示する
Call Stack	サブルーチン
Variables	変数の値を表示する
Modules	利用されているモジュールを表示する
Help Contents	ヘルプデータの一覧を表示する
Help Index	ヘルプの索引を表示する
Help Viewer	ヘルプの内容を表示する

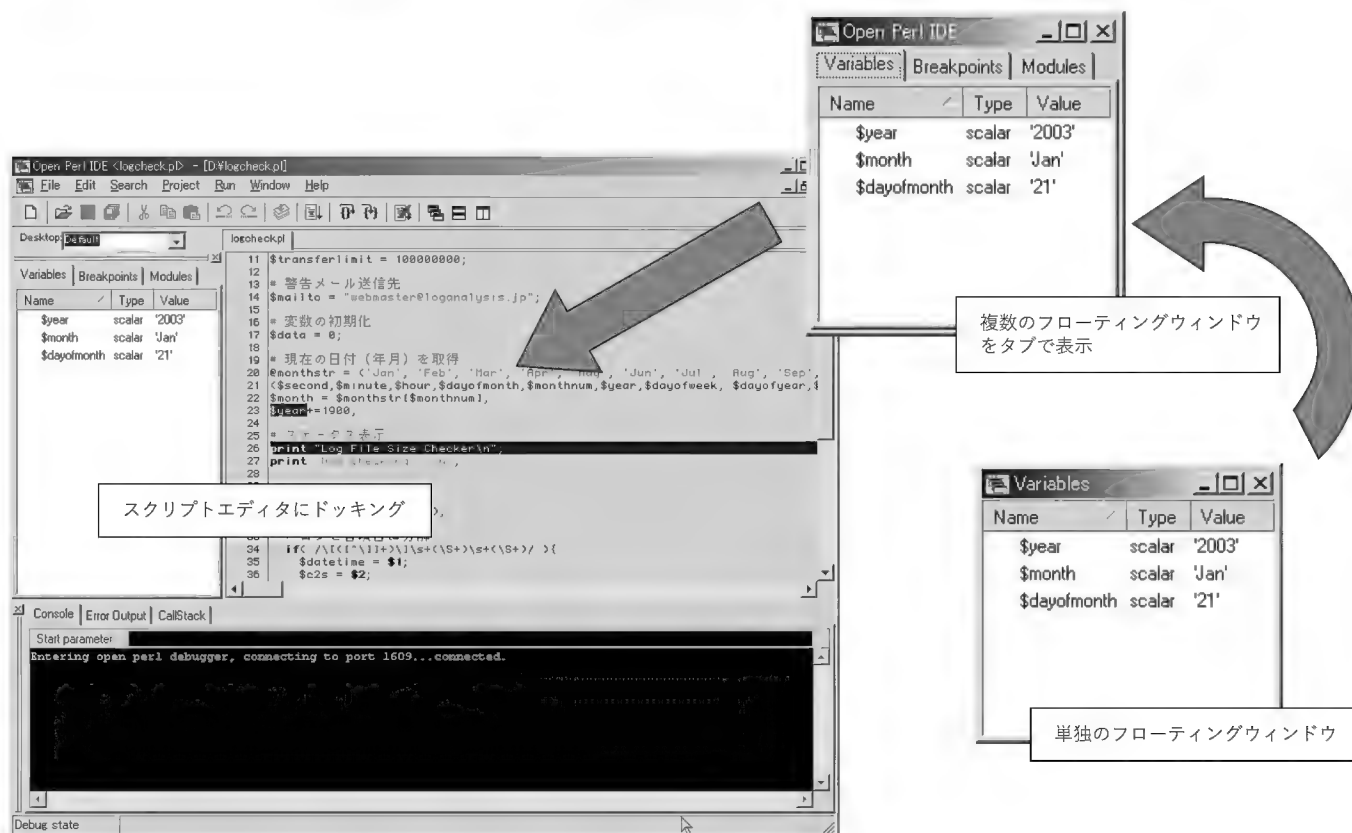
プログラミング

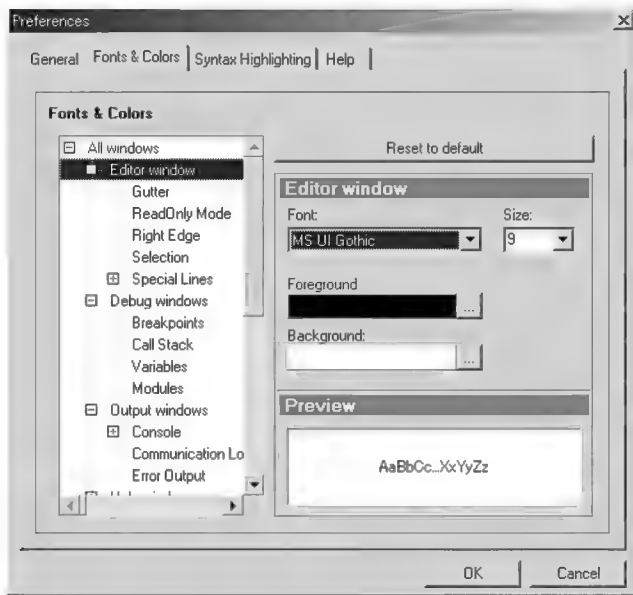
Open Perl IDEを利用したプログラミングの方法は，通常の

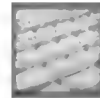
〔図2〕フローティング&ドッキング機能により，自由にウィンドウの配置が可能

複数のフローティングウィンドウをタブで表示

スクリプトエディタにドッキング

単独のフローティングウィンドウ

注3：このしくみは，ちょうどBorland社のDelphiのもつIDEと同じような機能である．そしてDelphiには，作成するアプリケーションにフローティング&ドッキングの機能をつけることができる．Open Perl IDEはDelphiで作られているため，同じようにフローティング&ドッキングの機能がついている．

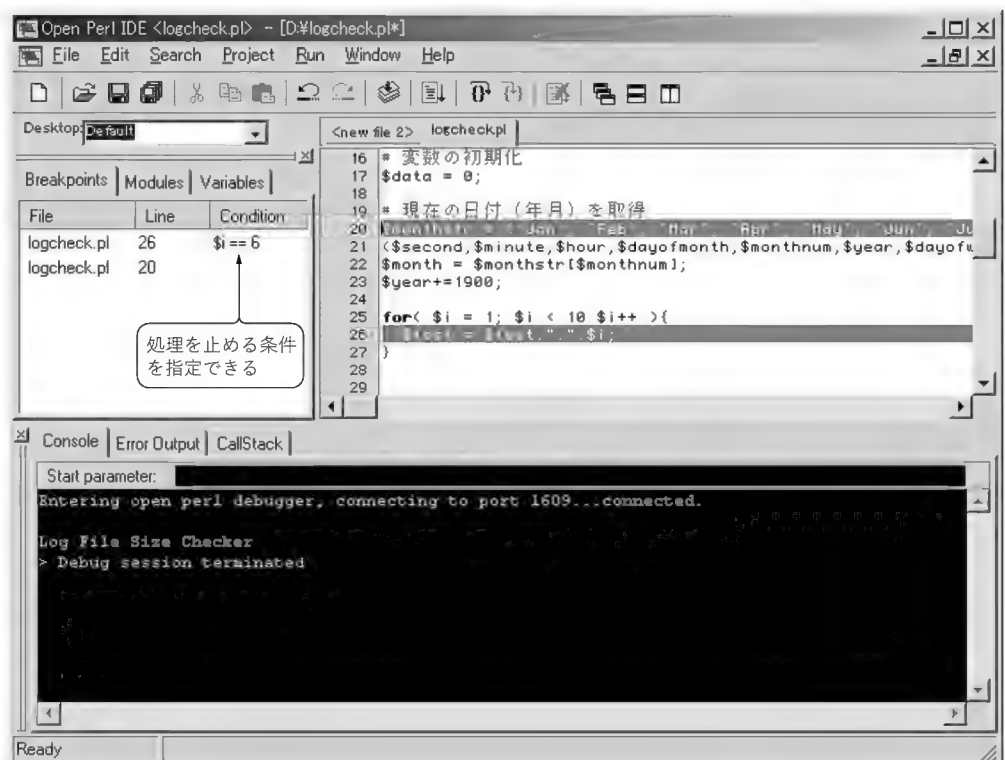
〔図3〕エディタで利用するフォントの指定

エディタと基本的には同じである．が，Perlのキーワード（命令や文字列，変数やコメントなど）がそれぞれ異なるスタイル（色，ボールド，イタリックなど）で表示されるようになっている（**図1**）．これは，統合開発環境をもつ言語ではよくある機能だが，これがあるだけでもソースコードの見やすさは格段に向上するうえ，スペルミスなども見つけやすくなるので，たいへん便利な機能である．各キーワードにどんなスタイルを割り当てるかは，設定ウィンドウで自由に変更することができる．

また，エディタは複数のファイルを同時に開くことができ，それらはタブで切り替えられるようになっている．

実行とデバッグ

プログラムが完成したら，続いて実行する．Open Perl IDEでは，もちろんプログラムをそのまま実行することができる．実行結果はConsoleウィンドウに出力される．スクリプトにパラメータを渡したい場合には，Consoleウィンドウ上部にある「Start parameter」というフィールドにデータを記述すればいい．もし，実行時にエラーが発生した場合は，エラーメッセージがError Outputウィンドウに出力される．

また，スクリプトに書式エラーがあった場合には，エラーの発生した行がハイライト表示されるようになっている．打ち間違いや勘違いによるスクリプト中のスペルミスは，少し長いプログラムを書いた場合は（少なくとも筆者の場合）必ずどこかに紛れ込んでしまうものなので，これは非常にありがたい機能である．

さて，プログラムのスペルミスがなくなって次に行うことは，プログラムが正しく動作するかどうかを確かめる作業である．ここでも，Open Perl IDEが力を発揮する．このIDEのもっとも大きな特徴は，デバッグに関する機能がIDEと結び付けられており，GUIを利用して，デバッグの機能が簡単に使えるようにしている点である．では，その機能をみていくことにしよう．

〔図4〕ブレークポイントの指定

〔図5〕実行中に変数の中身を表示できる

```
20 @monthstr = ('Jan', 'Feb', 'Ma
21 ($second,$minute,$hour,$dayofm
22 $month = $monthstr[$monthnum];
23 $year+=1900;
24
25 # ;$year: scalar = 2003
26 print "Log File Size Checker\n
27 print "Now Checking...\n";
28
29
30 # ログのチェック
```

● ブレークポイント

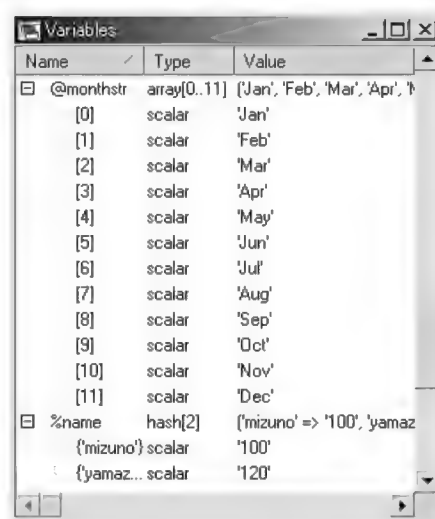
Open Perl IDEではプログラムにブレークポイントを設定することができる．ブレークポイントは，ソースコードエディタにおいて，行番号が表示されている部分をクリックすることで設定できる．設定されたブレークポイントは，Brakepointsウィンドウに一覧表示される．さらに，このBreakpointsウィンドウにはファイル名，行番号のほかに「Condition」という項目が用意されている．そこに変数の値などの条件を指定することで，その条件を満たす場合のみに，プログラムの実行が停止するように設定することも可能になっている(**図4**)．

● 変数

Open Perl IDEでは，一行ずつプログラムを実行するステップ実行も可能である．ステップ実行を行うか，ブレークポイントが設定されていた場合，プログラムはデバッグモードで実行されることになる．デバッグモードでプログラム実行中の場合，スクリプトエディタ上の変数名にマウスカーソルを近づけると，その変数の中身がツールチップとなって表示される(**図5**)．

また，Variablesウィンドウが用意されており，そこに変数名を入力すると，現在のその値が表示されるしくみになっていて，チェックしておきたい変数を常にこのウィンドウ上に表示したまま，デバッグ作業を行うことができる．ちなみに，ハッシュや配列を指定すると，すべての内容が階層表示されるようにもなっている(**図6**)．

なおPerlには，もともと次のように-dスイッチをつけて起動

〔図6〕Variablesウィンドウでは，配列，ハッシュは階層表示される

することで，標準のデバッグモードが起動するようになっている．

```
perl -d samplescript.pl
```

このデバッガはたいへん強力で便利なものだが，コマンドライン上で動作するものなので，取り扱うためにはデバッグ用のコマンドを覚える必要があった．しかし，Open Perl IDEを利用すれば，コマンドを覚えていなくても，GUIからデバッグが可能になる．

ちなみに，Perl標準のデバッグモードでスクリプトを実行した場合，「perl5db.pl」というデバッグ用のスクリプトが利用される．Open Perl IDEからデバッグモードでスクリプトを起動した場合，Perl標準の「perl5db.pl」ではなく，Open Perl IDEと同じディレクトリに置かれた「perl5db.pl」が利用される．このスクリプトは，Open Perl IDEと通信を行う機能をもつデバッグ用スクリプトである．このスクリプトを使うことで，Open Perl IDEはスクリプトとの連携を行うのだ．なお，Open Perl IDEとこのスクリプトの間での通信は，TCPのポート1538番を使って行われている．

Perlを始めよう！

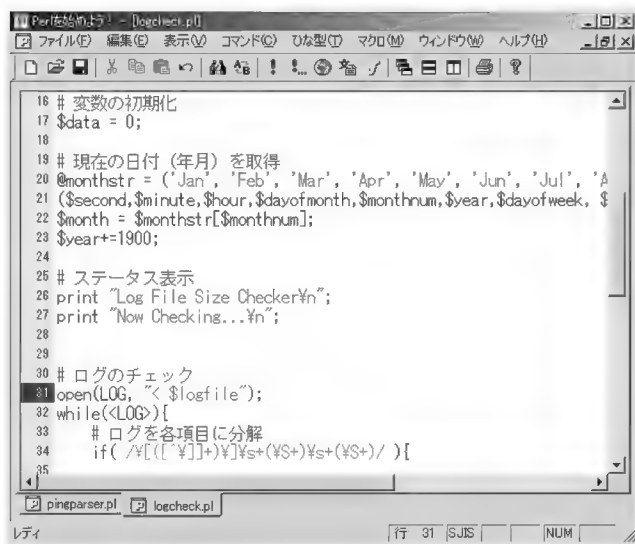
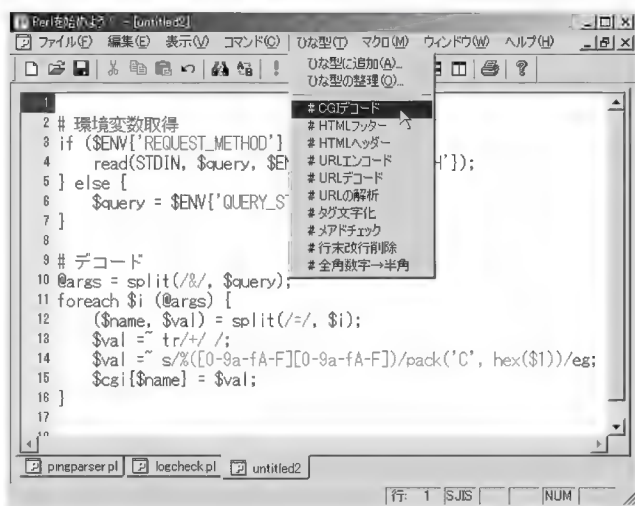
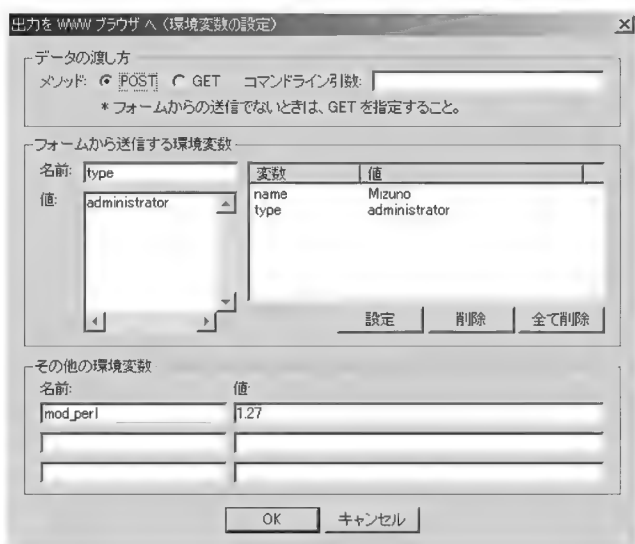
「Perlを始めよう！」は，青倉克浩(AOK)氏によって作られた，日本製のPerlの統合開発環境である．日本製だけあって，文字コードの問題や，JPerlへの対応など，日本語を利用するための配慮がなされているのが特徴である．

インストールと設定

「Perlを始めよう！」も，とくに特別なインストール作業の必要のないソフトウェアである．圧縮ファイルを伸張すれば，そのまま実行が可能である．ただし，こちらももちろん，Perlの実行環境自体はインストールされないので，別途インストール

DATA

名称：Perlを始めよう！
作者：青倉克浩(AOK)氏
Webサイト：http://hp.vector.co.jp/authors/VA010286/
現在のバージョン：Ver.2.0.3.6
ダウンロードサイズ：263Kバイト(LHA圧縮ファイル)
OS：Windows 95/98/NT4/2000/XP

〔図7〕「Perlを始めよう！」実行画面

〔図8〕ひな型を利用して決まりきったコードの入力を簡便化

〔図9〕フォームデータを作成し，CGIの実行をエミュレート

〔リスト1〕マクロはPerlで実装されている

```
# コメント化
while (<>) {
    s/¥s*$//;
    print "# $ ¥n";
}
```

する必要がある．

実行画面は**図7**のようになっている．基本的にはソースコードエディタのみのシンプルなウィンドウである．日本生まれの開発環境なので，もちろん何の設定も行わずに，日本語の入力/表示が可能になっている．

「Perlを始めよう！」でも，Open Perl IDEと同様，Perlの実行プログラムのファイルパスを設定ウィンドウで指定する必要がある．設定ウィンドウは「コマンド」メニューの「環境設定...」メニュー項目で表示される．設定ウィンドウでは，利用する実行プログラムが`Perl.exe`なのか，`JPerl.exe`[注4]なのかを指定することもできる．

プログラミング

「Perlを始めよう！」も，Open Perl IDEと同様，Perlのキーワードを色付きで表示してくれる機能がある(**図7**)．また，複数のスクリプトファイルをタブで切り替えるしくみも備わっている．

また，スクリプト読み込み時に文字コードを自動的に判別する機能(設定画面で機能を有効にする必要あり)もある．このあたりも，さすが日本生まれの開発環境といえるだろう．

さて，「Perlを始めよう！」には，「ひな型」および「マクロ」という二つのメニューがある．「ひな型」は，プログラム中に埋め込むことができるプログラムのパーツを登録しておき，好きなときにスクリプトに貼り付けることができる機能だ．メニューを開くと「CGIデコード」，「URLエンコード」など，CGIの作成時にかなりよく使う処理が並ぶ(**図8**)．

ひな型はアプリケーションディレクトリにある，「template」というディレクトリ内にそれぞれ単体のファイルとして保存されており，新たにひな型を追加することも可能である．

続いて「マクロ」には，「`print`文にする」，「インデントを下げる」など，ソースコードを加工するためのさまざまなツールが登録されている．これらのマクロは，実際にはそれぞれPerlで書かれたスクリプトとなっており，アプリケーションディレクトリにある，「macro」というディレクトリに保存されている．

リスト1に，登録されているマクロの一つである「コメント化」のソースを示す．マクロスクリプトは，選択された行を標準入力として受け取り，標準出力として変換したコードを出力するようになっている．自分で作成した新しいマクロを登録することも可能である．Perl向けの開発環境であるから，マクロとし

注4：日本語に対応したPerl．正規表現などで，日本語が1文字と認識される．

てPerlが利用できれば，マクロ用に新たな言語や書式を覚える必要がないわけで，非常に合理的であるといえる．

「Perlを始めよう！」では，「ひな型」と「マクロ」に，自分がよく利用する機能を追加していくことで，より開発しやすい環境へと鍛えていくことができるわけだ．

実行とデバッグ

「Perlを始めよう！」でスクリプトを実行すると，自動的にDOSプロンプトウィンドウが立ち上がり，スクリプトが実行される．また，コマンドメニューの「DOS窓を表示しない」にチェックを入れると，アウトプットウィンドウが開き，そこに実行結果が表示されるようになっている．ちなみにこのアウトプットウィンドウは，Open Perl IDEの各種フローティングウィンドウと同様，フローティングウィンドウとしても，またソースコードエディタにドッキングしても利用可能になっている．

また，「Perlを始めよう！」では，出力結果をWebブラウザに表示することもできる．メニューから「出力をwwwブラウザへ…」を選択すると，出力結果がテキストファイル(HTMLファイル)として出力され，Webブラウザが起動して，表示されるようになっている．しかも，これはCGIのチェックを目的とした機能なので，スクリプトに渡すフォームデータをエミュレートする機能がついている．実行時に**図9**のようなダイアログが表示され，データを入力するのだが，環境変数の値を自由に設定できたり，呼び出しに利用するHTTPメソッド(POSTとGET)を指定できる点など，よく考えられており，非常に便利にできていると感じた．

「Perlを始めよう！」にも，スクリプトをデバッグモードで起動する機能がついている．ただし，Open Perl IDEとは異なり，エディタとは連動していない．DOSプロンプトウィンドウが立ち上がって，Perlの標準のデバッグ機能が実行されるようになっている．

おわりに

今回紹介したPerlの開発環境は，どちらも非常に使いやすくて便利なツールである．しかし，この二つのIDEは，得意分野が若干異なっている．Open Perl IDEがデバッグの簡便性を高めていることが特徴的なIDEであったのに対し，「Perlを始めよう！」は「マクロ」や「ひな型」などの機能により，プログラミング中の簡便性を高めることを重視している開発環境であると筆者は考えている．また「Perlを始めよう！」の場合，日本語との親和性が高いことや，CGI向けの機能が多いことも，重要な特徴となっているといえるだろう．

どちらを使うのかは，好みや利用目的によって異なるが，Perlのプログラムの作成やデバッグなどの際に，少しでも「横着がしたい」と考える，筆者のような人間にはどちらもたいへんありがたいツールである．

また，こういったIDEは，プログラミングの初心者にとっても，非常に役立つツールといえるだろう．OSがGUIを搭載するのがあたりまえになった現在では，Perlのようにすべてをコマンドラインで行わなければならないツールは，非常に敷居が高く感じてしまう人も多いからだ．

筆者には大学で研究を続けている知り合いが何人かいるが，たまに統計解析ソフトウェアの結果や実験，調査の記録などのテキストデータを，別の形式に加工したいという相談を受けることがある．いちいち手ですべてを加工するのはたいへんなので，何とか自動化できないか，というわけである．それほど難しいものではないので，単機能なプログラムを組んであげたりしているのだが，そういったプログラムはえてして「使い捨て」というか，そのときにしか使えないものとなる．難しいプログラムではないので，筆者としてはそれほど苦ではないが，頼む側としては，何度も微妙に違うプログラムを頼むのも，気が引けるようだし，筆者も忙しくてなかなか手伝えないこともある．そういう相談を受けるたびに，Perlが使えれば，自分でぱっとスクリプトを書けばいいだけなのに，と感じる．

実際，Perlの利用をすすめたこともあるが，しかし，やはりコマンドラインでプログラムを実行し，デバッグし，といった作業はどうも苦手な人が多いような印象を受けた．しかし，そういった人たちでも，こういった使いやすいIDEがあれば，Perlを始めるにあたっての，抵抗が少なくなるのではないかと思う．ということで，筆者もこれを機会に，その友人たちにIDEの利用をすすめてみようと思っている．

みずの・たかあき

やり直しのための信号数学

第15回

FFTによる信号処理応用(システム設計編Ⅱ)

三谷政昭

前回は，FFTを用いたディジタルシステムを設計する事例(ディジタルフィルタ)を中心に，伝達関数(あるいはシステム関数)の近似設計におけるFFTの適用の仕方，考え方について解説した．

今回はFFTによるシステム設計の第2弾として，希望する周波数特性の等間隔にとった周波数サンプル値(設計仕様に相当)を満たすディジタルシステムを導き出す方法を紹介する．

まず，「周波数サンプリング」とよばれる設計法がDFT計算そのものであることを示し，本連載の第9回「FFT計算アルゴリズムの考え方」(2002年5月号)で述べた信号解析ディジタルフィルタの“周波数サンプル点ゼロ形特性”をたくみに利用したものであることに言及する．また，周波数特性をアダプティブ(adaptive：適応的)に変えられることを示す．(筆者)

ディジタルシステムの周波数スペクトル特性

いま，設計したいディジタルシステムの周波数スペクトル特性を$G(e^{j\omega T})$とするとき，周波数$(0 \leqq \omega T < 2\pi)$を等間隔にとった$N$個の周波数サンプル点，すなわち，

$$\omega_\ell T = \ell \times \frac{2\pi}{N} \;;\; \ell = 0,\ 1,\ 2,\ \ldots,\ (N-1) \quad \cdots\cdots(1)$$

に対するスペクトル値を，

$$G_\ell = G\left(e^{j\omega_\ell T}\right) = G\left(e^{j\ell\frac{2\pi}{N}}\right) \quad \cdots\cdots(2)$$

と表す(**図15.1**)．このとき，これらN個のすべての周波数サンプル点でスペクトル値が設計仕様に一致するようなディジタルシステムを得ようというわけである．

一方，ディジタルシステムに単位インパルス，すなわち，

$$x_k = \begin{cases} 1\,;\, k=0 \\ 0\,;\, k \neq 0 \end{cases} \quad \cdots\cdots(3)$$

を入力したときの応答出力(N個のサンプル値)を$\left\{h_k\right\}_{k=0}^{k=N-1}$とするとき，$z$変換して，

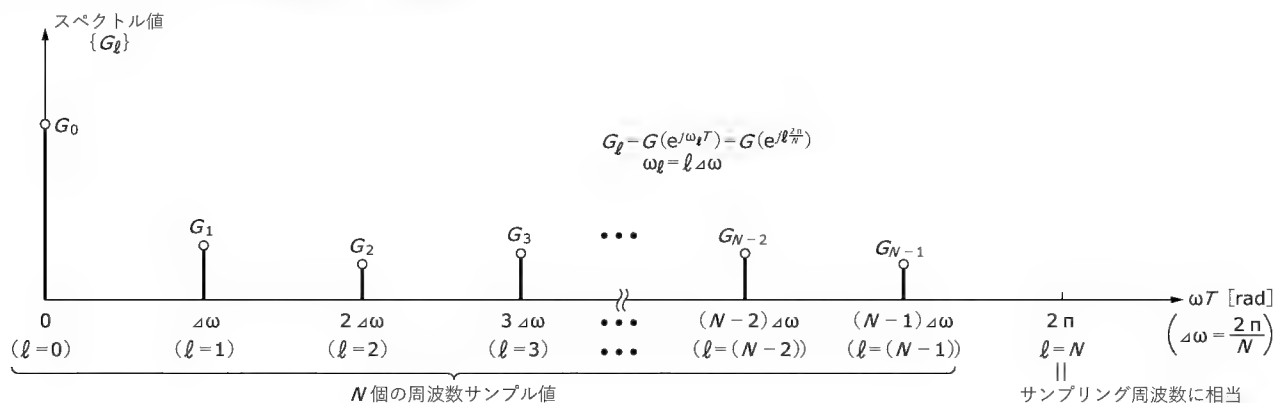
$$H(z) = h_0 + h_1 z^{-1} + h_2 z^{-2} + \ldots + h_{N-1} z^{-(N-1)} \quad \cdots\cdots(4)$$

となり，その周波数特性は$z = e^{j\omega T}$を代入することにより，

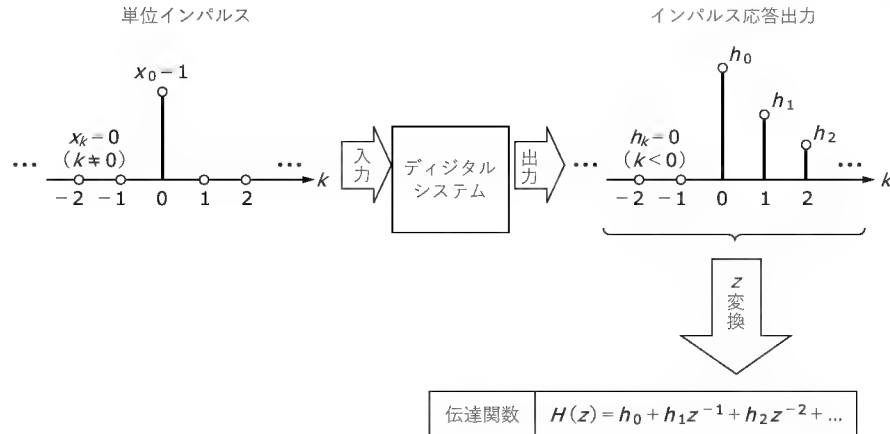
$$H\left(e^{j\omega T}\right) = h_0 + h_1 e^{-j\omega T} + h_2 e^{-j2\omega T} + \ldots + h_{N-1} e^{-j(N-1)\omega T} \quad \cdots\cdots(5)$$

と表される(**図15.2**)．

そこで，式(5)の式(1)の角周波数$\left\{\omega_\ell T\right\}_{\ell=0}^{\ell=N-1}$におけるサンプ

〔**図15.1**〕ディジタルシステムの周波数スペクトル特性

〔図 15.2〕
単位インパルス入力に対する
応答出力と伝達関数

単位インパルス　$x_0=1$,　$x_k=0$ $(k\neq 0)$　→ 入力 → ディジタルシステム → 出力 → インパルス応答出力　h_0, h_1, h_2,　$h_k=0$ $(k<0)$

z 変換 → 伝達関数　$H(z)=h_0+h_1z^{-1}+h_2z^{-2}+\ldots$

ル値，すなわち，

$$H_\ell=H\left(e^{j\ell\frac{2\pi}{N}}\right)\ ;\ \ell=0,\ 1,\ 2,\ \ldots,\ (N-1)\ \cdots\cdots(6)$$

について，

$$H_\ell=G_\ell\ ;\ \ell=0,\ 1,\ 2,\ \ldots,\ (N-1)\ \cdots\cdots(7)$$

となるように，ディジタルシステムの係数$\{h_k\}_{k=0}^{k=N-1}$を決定する問題に帰着される．式(2)〜式(7)に基づき，

$$W=e^{-j\frac{2\pi}{N}}\ \cdots\cdots(8)$$

とおけば，

$$\begin{cases}G_0=h_0+h_1+h_2+\ldots+h_{N-1}\\ G_1=h_0+h_1W+h_2W^2+\ldots+h_{N-1}W^{N-1}\\ G_2=h_0+h_1W^2+h_2W^4+\ldots+h_{N-1}W^{2(N-1)}\\ \quad\vdots\\ G_{N-1}=h_0+h_1W^{(N-1)}+h_2W^{2(N-1)}+\ldots+h_{N-1}W^{(N-1)(N-1)}\end{cases}\ \cdots\cdots(9)$$

のように，ディジタルシステムのN個の係数$\{h_k\}_{k=0}^{k=N-1}$に関する連立方程式として表される．

ところで，N個の係数$\{h_k\}_{k=0}^{k=N-1}$をディジタル信号とみなせば，そのDFT値は，

$$\begin{cases}F_0=\dfrac{1}{N}\{h_0+h_1+h_2+\ldots+h_{N-1}\}\\ F_1=\dfrac{1}{N}\{h_0+h_1W+h_2W^2+\ldots+h_{N-1}W^{N-1}\}\\ F_2=\dfrac{1}{N}\{h_0+h_1W^2+h_2W^4+\ldots+h_{N-1}W^{2(N-1)}\}\\ \quad\vdots\\ F_{N-1}=\dfrac{1}{N}\{h_0+h_1W^{(N-1)}+h_2W^{2(N-1)}+\ldots+h_{N-1}W^{(N-1)(N-1)}\}\end{cases}\ \cdots\cdots(10)$$

と表される．式(9)と式(10)とを比較すると，ちょうど$(1/N)$倍の関係，

$$F_\ell=\frac{1}{N}\times G_\ell\ ;\ \ell=0,\ 1,\ 2,\ \ldots,\ (N-1)\ \cdots\cdots(11)$$

が成立している．また，式(10)のIDFTは，

$$\begin{cases}h_0=F_0+F_1+F_2+\ldots+F_{N-1}\\ h_1=F_0+F_1W^{-1}+F_2W^{-2}+\ldots+F_{N-1}W^{-(N-1)}\\ h_2=F_0+F_1W^{-2}+F_2W^{-4}+\ldots+F_{N-1}W^{-2(N-1)}\\ \quad\vdots\\ h_{N-1}=F_0+F_1W^{-(N-1)}+F_2W^{-2(N-1)}+\ldots+F_{N-1}W^{-(N-1)(N-1)}\end{cases}\ \cdots\cdots(12)$$

であることから，式(11)を考慮して，周波数スペクトルの設計仕様$\{G_\ell\}_{\ell=0}^{\ell=N-1}$より，

$$\begin{cases}h_0=\dfrac{1}{N}\{G_0+G_1+G_2+\ldots+G_{N-1}\}\\ h_1=\dfrac{1}{N}\{G_0+G_1W^{-1}+G_2W^{-2}+\ldots+G_{N-1}W^{-(N-1)}\}\\ h_2=\dfrac{1}{N}\{G_0+G_1W^{-2}+G_2W^{-4}+\ldots+G_{N-1}W^{-2(N-1)}\}\\ \quad\vdots\\ h_{N-1}=\dfrac{1}{N}\{G_0+G_1W^{-(N-1)}+G_2W^{-2(N-1)}+\ldots+G_{N-1}W^{-(N-1)(N-1)}\}\end{cases}\ \cdots\cdots(13)$$

となる．さらに式(13)を式(5)に代入した後，設計仕様の周波数スペクトルのサンプル値$\{G_\ell\}_{\ell=0}^{\ell=N-1}$に着目して整理することにより，

$$\begin{aligned}H\left(e^{j\omega T}\right)=&G_0\left\{\frac{1+e^{-j\omega T}+e^{-j2\omega T}+\ldots+e^{-j(N-1)\omega T}}{N}\right\}\\ &+G_1\left\{\frac{1+W^{-1}e^{-j\omega T}+W^{-2}e^{-j2\omega T}+\ldots+W^{-(N-1)}e^{-j(N-1)\omega T}}{N}\right\}\\ &+G_2\left\{\frac{1+W^{-2}e^{-j\omega T}+W^{-4}e^{-j2\omega T}+\ldots+W^{-2(N-1)}e^{-j(N-1)\omega T}}{N}\right\}\\ &\quad\vdots\\ &+G_{N-1}\left\{\frac{1}{N}\left(1+W^{-(N-1)}e^{-j\omega T}+W^{-2(N-1)}e^{-j2\omega T}+\ldots+W^{-(N-1)(N-1)}e^{-j(N-1)\omega T}\right)\right\}\end{aligned}\ \cdots\cdots(14)$$

と表される．このとき，各周波数サンプル値に対する$\{\}$の中はそれぞれ，以下のように計算される．

$$\frac{1+e^{-j\omega T}+e^{-j2\omega T}+\ldots+e^{-j(N-1)\omega T}}{N}$$
$$=\frac{1+e^{-j\omega T}+\left(e^{-j\omega T}\right)^2+\ldots+\left(e^{-j\omega T}\right)^{(N-1)}}{N}$$
$$=\frac{1}{N}\times\frac{1-e^{-jN\omega T}}{1-e^{-j\omega T}}$$

$$\frac{1+W^{-1}e^{-j\omega T}+W^{-2}e^{-j2\omega T}+\ldots+W^{-(N-1)}e^{-j(N-1)\omega T}}{N}$$
$$=\frac{1+W^{-1}e^{-j\omega T}+\left(W^{-1}e^{-j\omega T}\right)^2+\ldots+\left(W^{-1}e^{-j\omega T}\right)^{(N-1)}}{N}$$
$$=\frac{1}{N}\times\frac{1-\left(W^{-1}e^{-j\omega T}\right)^N}{1-W^{-1}e^{-j\omega T}}=\frac{1}{N}\times\frac{1-W^{-N}e^{-jN\omega T}}{1-W^{-1}e^{-j\omega T}}$$
$$=\frac{1}{N}\times\frac{1-e^{-jN\omega T}}{1-W^{-1}e^{-j\omega T}}$$

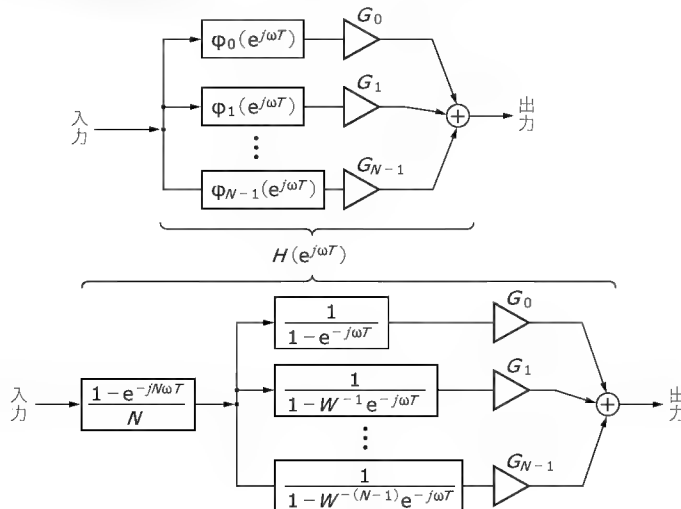
$$\frac{1+W^{-2}e^{-j\omega T}+W^{-4}e^{-j2\omega T}+\ldots+W^{-2(N-1)}e^{-j(N-1)\omega T}}{N}$$
$$=\frac{1+W^{-2}e^{-j\omega T}+\left(W^{-2}e^{-j\omega T}\right)^2+\ldots+\left(W^{-2}e^{-j\omega T}\right)^{(N-1)}}{N}$$
$$=\frac{1}{N}\times\frac{1-\left(W^{-2}e^{-j\omega T}\right)^N}{1-W^{-2}e^{-j\omega T}}=\frac{1}{N}\times\frac{1-W^{-2N}e^{-jN\omega T}}{1-W^{-2}e^{-j\omega T}}$$
$$=\frac{1}{N}\times\frac{1-e^{-jN\omega T}}{1-W^{-2}e^{-j\omega T}}$$

$$\vdots$$

以下，同様にして計算することにより，式(14)は，

$$H\left(e^{j\omega T}\right)=G_0\phi_0\left(e^{j\omega T}\right)+G_1\phi_1\left(e^{j\omega T}\right)+G_2\phi_2\left(e^{j\omega T}\right)+\ldots+G_{N-1}\phi_{N-1}\left(e^{j\omega T}\right) \quad\cdots\cdots\cdots(15)$$

ただし，$\phi_\ell\left(e^{j\omega T}\right)=\frac{1}{N}\times\frac{1-e^{-jN\omega T}}{1-W^{-\ell}e^{-j\omega T}}$ ……………(16)

〔図 15.3〕周波数スペクトル $H(e^{j\omega T})$ のシステム合成

となる(**図 15.3**)．計算に際しては，

$$W^{-N}=\left(e^{-j\frac{2\pi}{N}}\right)^{-N}=e^{j2\pi}=1 \quad\cdots\cdots\cdots(17)$$

$$1+\alpha+\alpha^2+\ldots+\alpha^{(N-1)}=\frac{1-\alpha^N}{1-\alpha} \quad\cdots\cdots\cdots(18)$$

の関係を利用している．

周波数スペクトル値と周波数解析ディジタルフィルタ

たとえばN=8として，式(15)，式(16)で表される周波数スペクトル値を実現するディジタルシステムの伝達関数を考えてみることにしよう．まず，式(16)において $\ell=0$ として，

$$\phi_0\left(e^{j\omega T}\right)=\frac{1}{8}\times\frac{1-e^{-j8\omega T}}{1-W^0e^{-j\omega T}}=\frac{1}{8}\times\frac{1-e^{-j8\omega T}}{1-e^{-j\omega T}} \quad\cdots\cdots(19)$$

であり，分母と分子がそれぞれ，

$$1-e^{-j8\omega T}=e^{-j4\omega T}\left(e^{j4\omega T}-e^{-4\omega T}\right) \quad\cdots\cdots\cdots(20)$$

$$1-e^{-j\omega T}=e^{-j\frac{\omega T}{2}}\left(e^{j\frac{\omega T}{2}}-e^{-\frac{\omega T}{2}}\right) \quad\cdots\cdots\cdots(21)$$

と変形される．ここで，

$$e^{j\theta}-e^{-j\theta}=j2\sin\theta \quad\cdots\cdots\cdots(22)$$

となる関係より，式(20)，式(21)はそれぞれ，

$$1-e^{-j8\omega T}=e^{-j4\omega T}\left\{j2\sin(4\omega T)\right\}$$

$$1-e^{-j\omega T}=e^{-j\frac{\omega T}{2}}\left\{j2\sin\left(\frac{\omega T}{2}\right)\right\}$$

と表されることから，最終的に式(19)は，

$$\phi_0\left(e^{j\omega T}\right)=\frac{1}{8}\times\frac{e^{-j4\omega T}\left\{j2\sin(4\omega T)\right\}}{e^{-j\omega T/2}\left\{j2\sin\left(\frac{\omega T}{2}\right)\right\}}$$
$$=e^{-j7\omega T/2}\frac{\sin(4\omega T)}{8\sin\left(\frac{\omega T}{2}\right)} \quad\cdots\cdots\cdots(23)$$

となる．式(23)において振幅特性$|\phi_0(e^{j\omega T})|$を考えると，

$$\left|\phi_0\left(e^{j\omega T}\right)\right|=\underbrace{\left|e^{-j7\omega T/2}\right|}_{1}\left|\frac{\sin(4\omega T)}{8\sin\left(\frac{\omega T}{2}\right)}\right|=\left|\frac{\sin(4\omega T)}{8\sin\left(\frac{\omega T}{2}\right)}\right| \quad\cdots\cdots(24)$$

と表される．ここで$\sin(4\omega T)=0$，すなわちゼロ(零)点となる周波数は，$4\omega T=\pi$，2π，3π，4πより，

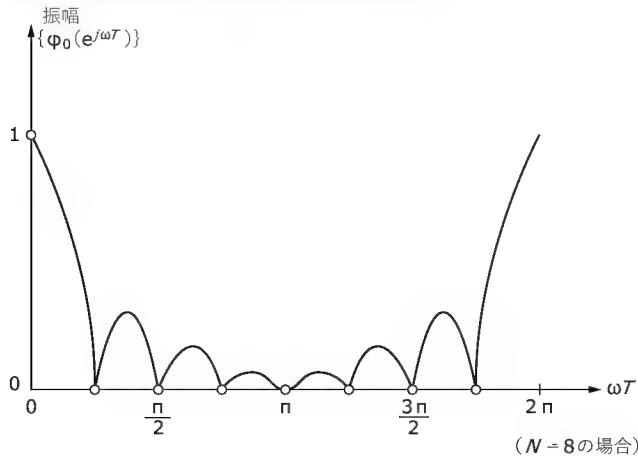
$$\omega T=\frac{\pi}{4},\ \frac{2\pi}{4},\ \frac{3\pi}{4},\ \pi$$

となる．また，$\omega T=0$のときは，ロピタルの定理，すなわち，

$$\lim_{x\to0}\frac{g(x)}{f(x)}=\lim_{x\to0}\frac{\frac{d}{dx}\{g(x)\}}{\frac{d}{dx}\{f(x)\}} \quad\cdots\cdots\cdots(25)$$

を，$x=\omega T$，$g(x)=\sin(4x)$，$f(x)=\sin(x)$として式(24)に適用すれば，

〔図15.4〕周波数サンプル点ゼロ形特性[$\varphi_0(e^{j\omega T})$]

〔図15.5〕周波数サンプル点ゼロ形特性の考え方

$$\lim_{\omega T\to 0}\frac{\sin(4\omega T)}{8\sin\left(\frac{\omega T}{2}\right)}=\lim_{\omega T\to 0}\frac{4\cos(4\omega T)}{8\times\frac{1}{2}\cos\left(\frac{\omega T}{2}\right)}=1 \quad\cdots\cdots\cdots(26)$$

となることから，**図15.4**のように“周波数サンプル点ゼロ形特性”を有することがわかる(2002年5月号「FFT計算アルゴリズムの考え方」を参照).

また，式(19)において$z=e^{j\omega T}$を考慮することにより，

$$\phi_0(z)=\frac{1}{8}\times\frac{1-z^{-8}}{1-z^{-1}}\ ;\ \ell=0,\ 1,\ 2,\ \ldots,\ 7\cdots\cdots\cdots(27)$$

と表される．さらに，式(27)を因数分解して整理すると，

$$\begin{aligned}\phi_0(z)&=\frac{1}{8}\times\frac{(1+z^{-4})(1+z^{-2})(1+z^{-1})(1-z^{-1})}{1-z^{-1}}\\&=\frac{1}{8}\times(1+z^{-4})(1+z^{-2})(1+z^{-1})\\&=\frac{1+z^{-4}}{2}\times\frac{1+z^{-2}}{2}\times\frac{1+z^{-1}}{2}\end{aligned}\quad\cdots\cdots\cdots(28)$$

となり，三つのLPFの組み合わせであることがわかる(**図15.5**, 2002年5月号「FFT計算アルゴリズムの考え方」を参照)．同様に，

$$\phi_\ell(e^{j\omega T})=\frac{1}{8}\times\frac{1-e^{-j8\omega T}}{1-W^{-\ell}e^{-j\omega T}}\ ;\ \ell=0,\ 1,\ 2,\ \ldots,\ 7\cdots(29)$$

の各周波数特性についても，

$$\phi_\ell(e^{j\omega T})=\begin{cases}1\ ;\ \omega T=\ell\times\left(\frac{2\pi}{8}\right)\\0\ ;\ \omega T\neq\ell\times\left(\frac{2\pi}{8}\right)\end{cases}\quad\cdots\cdots\cdots(30)$$

であることから，“周波数サンプル点ゼロ形特性”を有するわけで，FFTによるDFT値の計算自体が信号解析ディジタルフィルタに相当するのである(**図15.6**，**図15.7**)．なお，一般的なサンプル数Nに対しても，

$$\phi_\ell(e^{j\omega T})=\begin{cases}1\ ;\ \omega T=\ell\times\left(\frac{2\pi}{N}\right)\\0\ ;\ \omega T\neq\ell\times\left(\frac{2\pi}{N}\right)\end{cases}\quad\cdots\cdots\cdots(31)$$

となる関係が成立し，$z=e^{j\omega T}$を考慮することにより，

$$\phi_\ell(z)=\frac{1}{N}\times\frac{1-z^{-N}}{1-W^{-\ell}z^{-1}}\quad\cdots\cdots\cdots(32)$$

と表される．

DFT値と周波数サンプリングシステム

ところで，式(32)の“周波数サンプル点ゼロ形特性”を有する関数$\left\{\phi_\ell(z)\right\}_{\ell=0}^{\ell=N-1}$を用いると，式(15)は$z=e^{j\omega T}$として，

$$H(z)=G_0\phi_0(z)+G_1\phi_1(z)+G_2\phi_2(z)+\ldots G_{N-1}\phi_{N-1}(z)\cdots(33)$$

と表される．ここで，式(33)に式(32)を代入して，共通項を括りだすと，

$$H(z)=\frac{1-z^{-N}}{N}\left[G_0H_0(z)+G_1H_1(z)+G_2H_2(z)+\ldots+G_{N-1}H_{N-1}(z)\right]\cdots(34)$$

ただし，$H_\ell(z)=\dfrac{1}{1-W^{-\ell}z^{-1}}$

となり，**図15.8**に示す構成のディジタルシステムとして実現できる．式(34)によるディジタルシステムの係数$\left\{G_\ell\right\}_{\ell=0}^{\ell=N-1}$は複素数なので，実数係数の構成になってはいない．

ところで，式(8)と式(17)より，

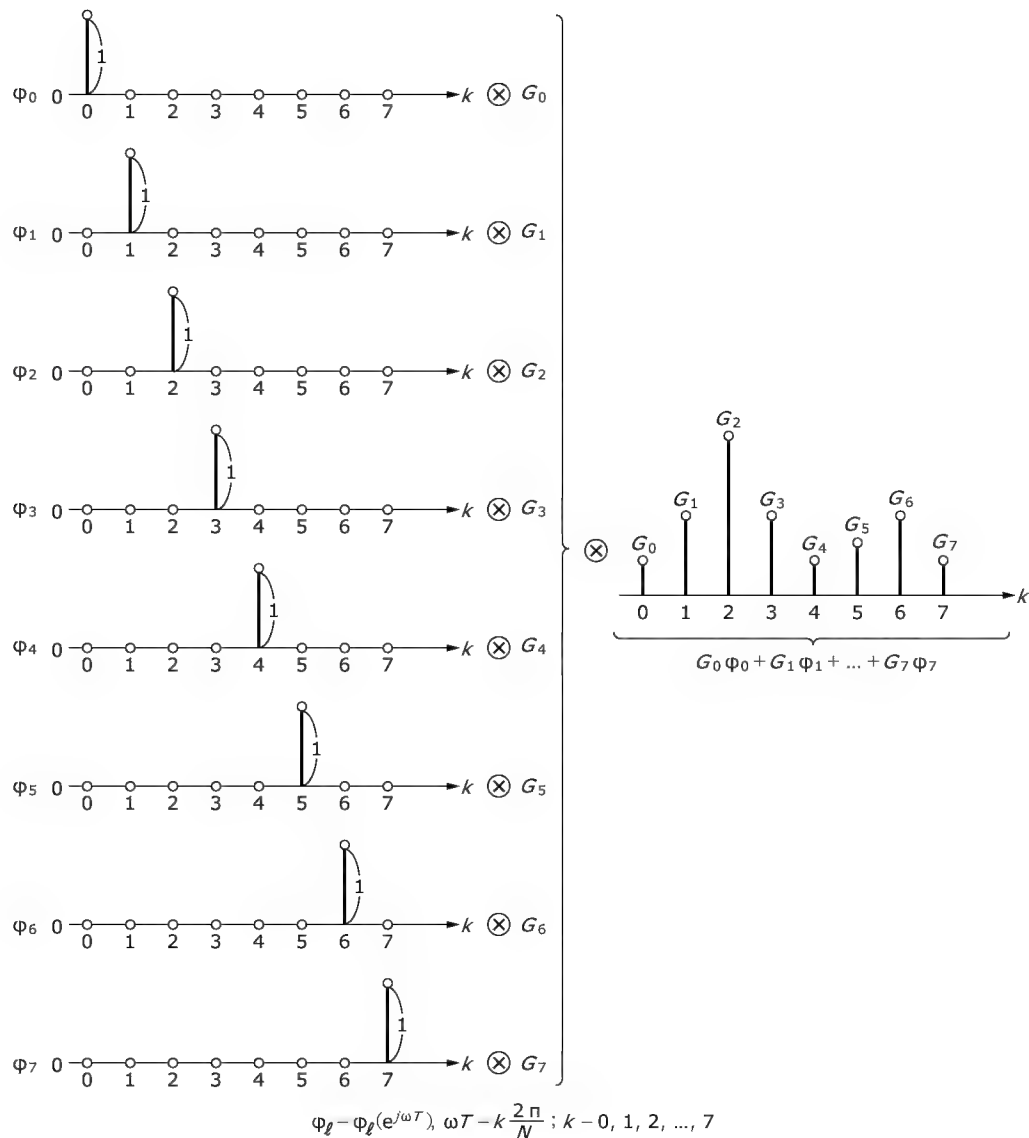
$$W^{-(N-\ell)}=W^\ell=e^{-j\ell\frac{2\pi}{N}}\quad\cdots\cdots\cdots(35)$$

であり，式(34)の複素係数ディジタルシステムを実数係数とするための条件として，

$$G_\ell=\overline{G_{N-\ell}}\quad\cdots\cdots\cdots(36)$$

を付与すればよい．たとえば，振幅特性である絶対値$\left\{G_\ell\right\}_{\ell=0}^{\ell=N-1}$は，

$$|G_\ell|=|G_{N-\ell}|\quad\cdots\cdots\cdots(37)$$

〔図 15.6〕
周波数サンプル点ゼロ形特性によるシステム合成

とし，位相特性である偏角 $\left\{\arg\left(G_\ell\right)\right\}_{\ell=0}^{\ell=N-1}$ は，

$$\arg\left(G_\ell\right) = -\arg\left(G_{N-\ell}\right) = \theta_\ell = \ell\pi \quad \cdots\cdots\cdots\cdots\cdots\cdots (38)$$

としてみよう（**図 15.9**）．

このとき，一般性を損なうことなくサンプル数 N を偶数とすると，式(34)は次のように変形できる．

$$H(z) = \frac{1-z^{-N}}{N}\Big[\left|G_0\right|H_0(z) - \left|G_1\right|H_1(z) + \left|G_2\right|H_2(z) - \ldots + (-1)^{N/2}\left|G_{N/2}\right|H_{N/2}(z)\Big] \quad \cdots\cdots\cdots\cdots (39)$$

ただし，

$$H_0(z) = \frac{1}{1-z^{-1}}$$

$$H_\ell(z) = \frac{2\left\{1-z^{-1}\cos\left(\ell\dfrac{2\pi}{N}\right)\right\}}{1-2z^{-1}\cos\left(\ell\dfrac{2\pi}{N}\right)+z^{-2}} \;;\; \ell = 1,\ 2,\ \ldots,\ \left(\frac{N}{2}-1\right)$$

$$H_{N/2}(z) = \frac{1}{1+z^{-1}}$$

式(39)に基づくディジタルシステムは**図 15.10** のように構成でき，周波数サンプリングシステムとよばれる．周波数サンプリングシステムでは，離散的な周波数サンプル点における振幅値（入力信号に対する出力信号の割合）を直接コントロールできるわけで，アダプティブ（適応的）なシステムに利用されている．

例題1

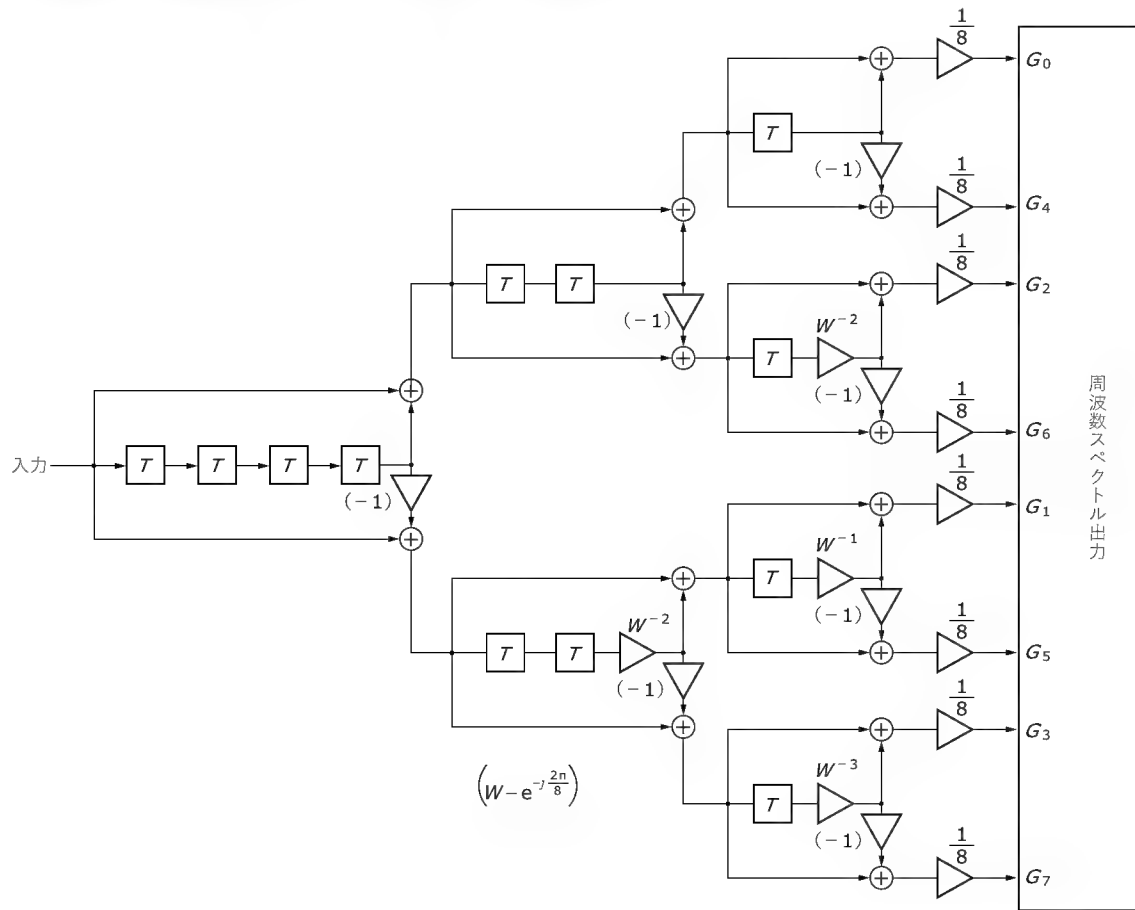
いま，N=4 として各周波数サンプル点で，

$|G_0|=2,\quad |G_1|=|G_3|=3,\quad |G_2|=4$

となる振幅値を有する周波数サンプリングシステムを構成し，$\omega T = 0,\ \dfrac{\pi}{2},\ \pi$ における振幅値がそれぞれ 2，3，4 になることを調べよ．

解答1

式(39)より，N=4 として，伝達関数は，

〔図15.7〕FFT計算に基づく信号解析ディジタルフィルタのブロック構成(N=8)

$$H(z)=\frac{1-z^{-4}}{4}\left[\frac{2}{1-z^{-1}}-\frac{6}{1+z^{-2}}+\frac{4}{1+z^{-1}}\right] \cdots\cdots\cdots (40)$$

であり，**図15.11**のように構成される．また，$\omega T=0,\ \frac{\pi}{2},\ \pi$ はそれぞれ，

$z=e^{j\omega T}=1,\ j,\ (-1)$

に対応するので，式(40)に代入して絶対値を計算すれば，各周波数における振幅値が求まる(**図15.12**)．ここで，式(40)を計算しやすいように，

$$H(z)=\frac{1+z^{-1}+z^{-2}+z^{-3}}{2}-\frac{3(1-z^{-2})}{2}+(1-z^{-1}+z^{-2}-z^{-3})$$

と変形し，得られた結果を以下に記す．

$z=1\quad(\omega T=0)\quad\rightarrow\quad|H(1)|=2=|G_0|$

$z=j\quad(\omega T=\pi/2)\quad\rightarrow\quad|H(j)|=3=|G_1|=|G_3|$

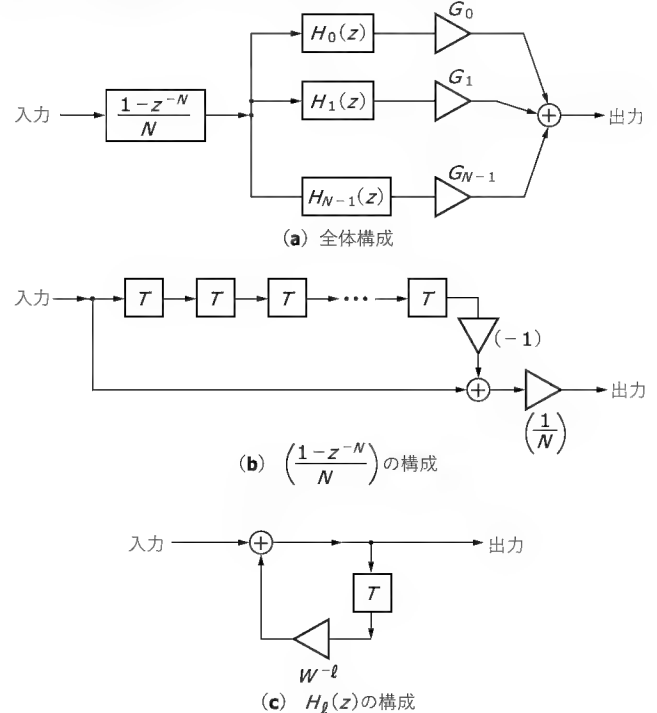
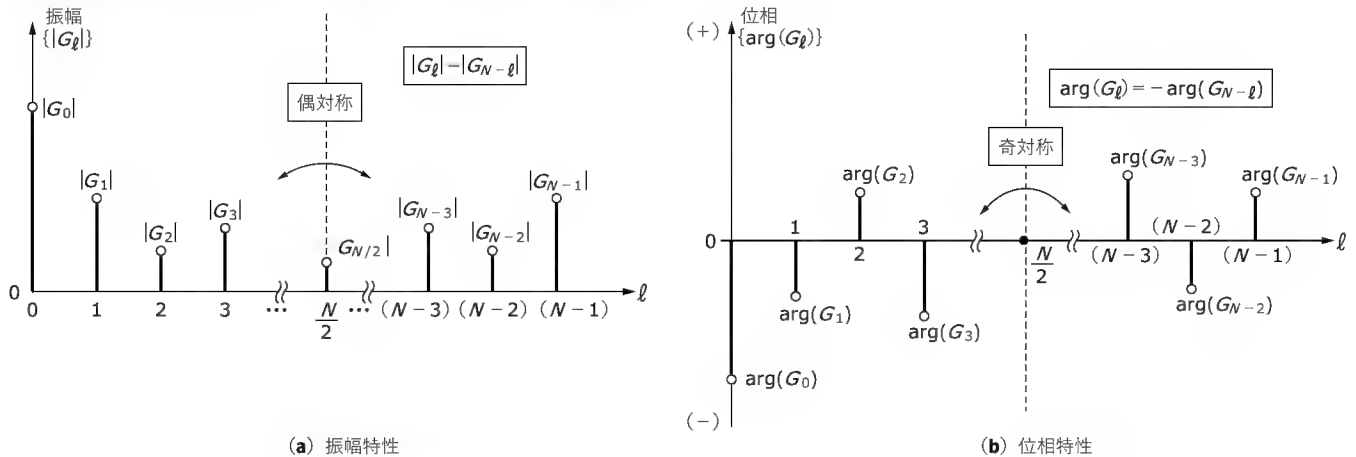
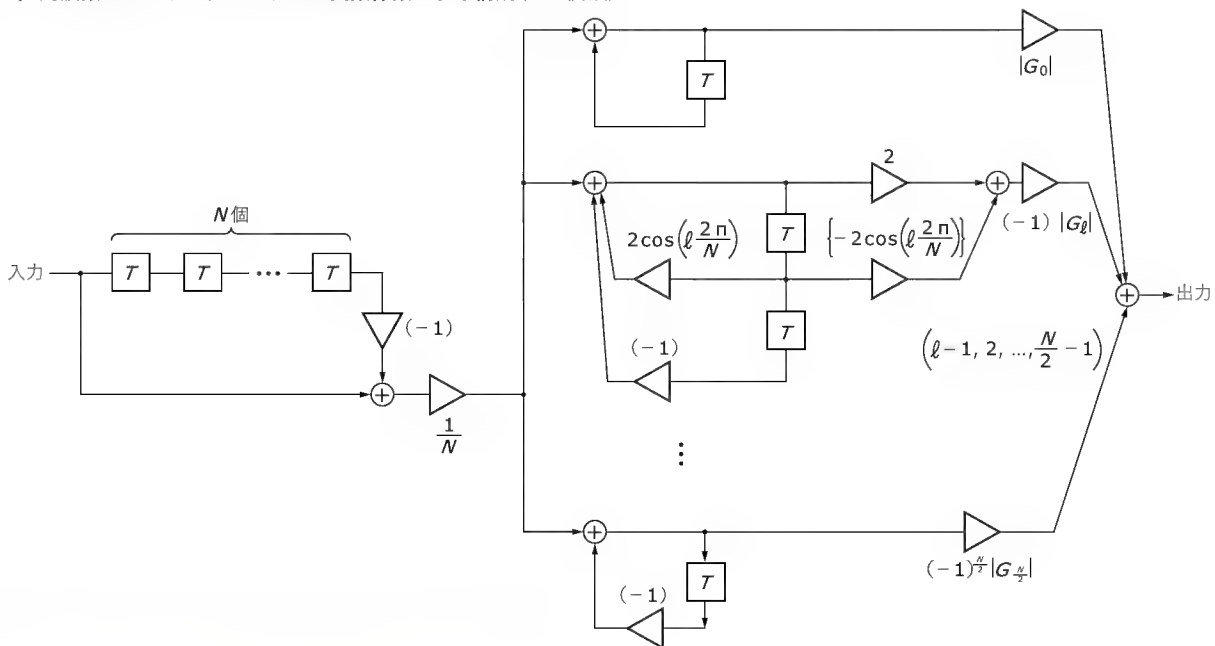
$z=-1\quad(\omega T=\pi)\quad\rightarrow\quad|H(-1)|=4=|G_2|$

よって，周波数サンプリングシステムでは離散的な周波数サンプル点の振幅値を直接コントロールできることが理解される．

〔図15.8〕周波数サンプリングシステムの構成

〔図 15.9〕実数係数を有するディジタルシステムの条件（N：偶数）

(a) 振幅特性

(b) 位相特性

〔図 15.10〕周波数サンプリングシステムの実数係数による構成（N：偶数）

ディジタルシステムの伝達関数の対数とDFT値

伝達関数$H(z)$の自然対数（底はeで，e=2.718 …）を採ったもの，すなわち，

$$\ln\{H(z)\} \quad \cdots\cdots (41)$$

をテイラー展開したときの係数が，希望する周波数特性$G(e^{j\omega t})$の対数，すなわち，

$$\ln\{G(e^{j\omega T})\} \quad \cdots\cdots (42)$$

のフーリエ級数の展開係数と等しくなるようにして，ディジタルシステムの伝達関数$H(z)$の係数を決める設計方法があり，ジョンソンの設計法とよばれている．

いま，ディジタルシステムの伝達関数$H(z)$が，

$$H(z) = H_0 \frac{1 + c_1 z^{-1} + c_2 z^{-2} + \ldots + c_M z^{-M}}{1 + d_1 z^{-1} + d_2 z^{-2} + \ldots + d_N z^{-N}} \quad \cdots\cdots (43)$$

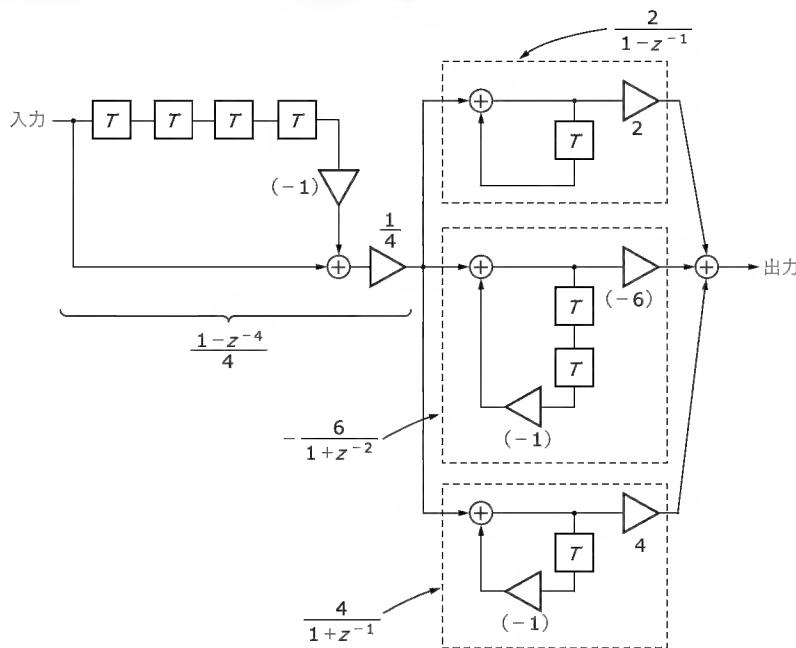
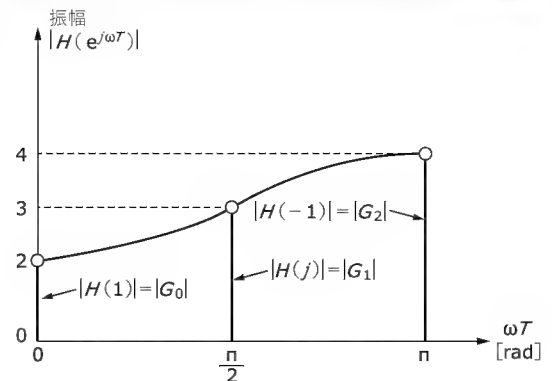
$$= H_0 \frac{(1-\alpha_1 z^{-1})(1-\alpha_2 z^{-1})\ldots(1-\alpha_M z^{-1})}{(1-\beta_1 z^{-1})(1-\beta_2 z^{-1})\ldots(1-\beta_N z^{-1})} \quad \cdots\cdots (44)$$

で表されるものとする．このとき，式(44)の対数を採ることにより，積（かけ算）が和（たし算）に，商（わり算）が差（ひき算）になることから，

$$H(z) = \ell n(H_0) + \ell n(1-\alpha_1 z^{-1}) + \ell n(1-\alpha_2 z^{-1}) + \ldots - \ell n(1-\beta_1 z^{-1}) - \ell n(1-\beta_2 z^{-1}) + \ldots \quad \cdots (45)$$

と表される．さらに，ある関数$p(x)$のテイラー級数展開式は，

$$p(x) = p(0) + \frac{p^{(1)}(0)}{1!}x + \frac{p^{(2)}(0)}{2!}x^2 + \frac{p^{(3)}(0)}{3!}x^3 + \ldots \quad \cdots\cdots (46)$$

〔図15.11〕ディジタルシステムの構成 例題1

〔図15.12〕ディジタルシステムの振幅周波数特性 例題2

ただし，$n!=1\times 2\times 3\times \ldots \times (n-1)\times n$

$$p^{(n)}(0)=\left.\frac{d^n p(x)}{dx^n}\right|_{x=0}$$

であり，

$p(x)=\ln(1-\lambda x)$ ；λは定数 ……………………(47)

に対しては，

$$p(0)=\ln(1-\lambda\times 0)=\ln(1)=0$$

$$p^{(n)}(0)=-\left.\frac{(n-1)\lambda^n}{(1-\lambda x)^n}\right|_{x=0}=-(n-1)!\lambda^n \;;\; n\geqq 1$$

となる関係より，$0!=1$を考慮して，

$$\ln(1-\lambda x)=-\lambda x-\frac{\lambda^2}{2}x^2-\frac{\lambda^3}{3}x^3-\ldots \quad \cdots\cdots(48)$$

とテイラー級数で展開される．

よって，式(48)を式(45)に適用すれば，

$$\ln\{H(z)\}=h_0+h_1z^{-1}+h_2z^{-2}+\ldots+h_mz^{-m}+\ldots \quad \cdots\cdots(49)$$

ただし，$h_0=\ell n(H_0)$ ……………………(50)

$h_m=f_m-g_m$ ；$m\geqq 1$ ……………………(51)

$$f_m=-\frac{1}{m}\sum_{k=1}^{N}(\alpha_k)^m \quad \cdots\cdots(52)$$

$$g_m=-\frac{1}{m}\sum_{k=1}^{N}(\beta_k)^m \quad \cdots\cdots(53)$$

と表される．

一方，対数で表した伝達関数$\ln\{H(z)\}$のテイラー展開係数$\{h_m\}_{m=0}^{m=\infty}$を，希望する対数で表した周波数特性$\ln\{G(e^{j\omega T})\}$のフーリエ展開係数に選べば，ディジタルシステムを設計することができる．このとき，対数周波数特性$\ln\{G(e^{j\omega T})\}$の周波数サンプル値$\{\ln(G_\ell)\}_{\ell=0}^{\ell=N-1}$のIDFT値として，

$$h_m=\frac{1}{N}\sum_{\ell=0}^{N-1}\ln(G_\ell)W^{-\ell m} \;;\; m=0,\ 1,\ 2,\ \ldots,\ (N-1) \cdots(54)$$

ただし，$G_\ell=G(e^{j\omega_\ell T})=G\left(e^{j\ell\frac{2\pi}{N}}\right)$ ；$\ell=0,\ 1,\ 2,\ \ldots,\ (N-1)$

で与えられる．

式(54)によって希望特性に対する$\{h_m\}_{m=0}^{m=N-1}$が求まると，式(50)～式(53)からゼロ点$\{\alpha_k\}_{k=1}^{k=M}$，極$\{\beta_k\}_{k=1}^{k=N}$が計算できることになり，ディジタルシステムの伝達関数が得られる．

たとえば，求めるディジタルシステムの伝達関数が分子のみ($d_m=0$ ；$m=1,\ 2,\ \ldots,\ N$)の場合，すなわち，

$g_m=0$ ($\beta_k=0$ ；$k=1,\ 2,\ \ldots,\ N$) ……………………(55)

を考えてみよう．このようなシステムでは，式(51)より，

$h_m=f_m$ ……………………(56)

であるので，式(52)から，$m\geqq 1$に対して，

$$mc_m=\sum_{k=0}^{m-1}(m-k)h_{m-k}c_k \;;\; c_0=1 \quad \cdots\cdots(57)$$

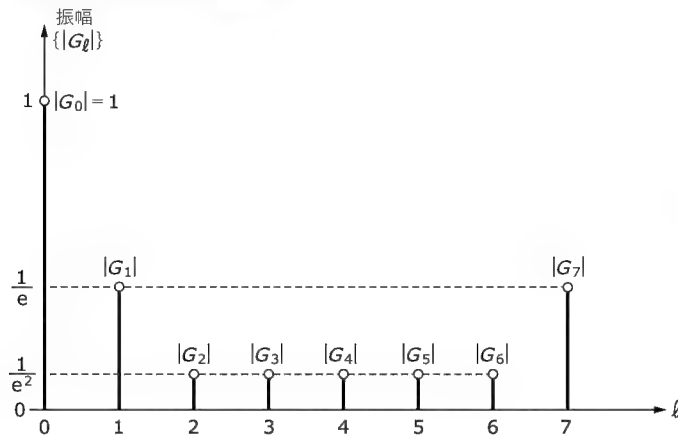
の関係が得られ，この関係式から伝達関数の係数$\{c_m\}_{m=1}^{m=M}$が計算される．なお，分子のみからなるディジタルシステムは“非再帰形(あるいは非巡回形)；non-recursive type”とよばれる．

同様に，求めるディジタルシステムの伝達関数が分母のみ($c_m=0$ ；$m=1,\ 2,\ \ldots,\ M$)の場合，すなわち，

$f_m=0$ ($\alpha_k=0$ ；$k=1,\ 2,\ \ldots,\ M$) ……………………(58)

を考えてみよう．このようなシステムでは，式(51)より，

$h_m=-g_m$ ……………………(59)

〔図15.13〕設計仕様 例題3

であるので，式(53)から，$m \geqq 1$ に対して，

$$md_m = -\sum_{k=0}^{m-1}(m-k)h_{m-k}d_k \;;\; d_0 = 1 \quad \cdots\cdots\cdots\cdots\cdots (60)$$

の関係が得られ，この関係式から伝達関数の係数 $\left\{d_m\right\}_{m=1}^{m=N}$ が計算される．なお，分子が1で分母のみからなるディジタルシステムは“純再帰形；purely recursive type”とよばれる．

例題2

いま，伝達関数 $H(z)$ の根を $z=\alpha,\ \beta,\ \gamma,\ \delta$ として，

$$H(z)=(1-\alpha z^{-1})(1-\beta z^{-1})(1-\gamma z^{-1})(1-\delta z^{-1})$$

において，

$$\begin{cases} p_1 = \alpha+\beta+\gamma+\delta = 10 \\ p_2 = \alpha^2+\beta^2+\gamma^2+\delta^2 = 30 \\ p_3 = \alpha^3+\beta^3+\gamma^3+\delta^3 = 100 \\ p_4 = \alpha^4+\beta^4+\gamma^4+\delta^4 = 354 \end{cases}$$

であるとき，伝達関数 $H(z)$ を，

$$H(z)=1+c_1z^{-1}+c_2z^{-2}+c_3z^{-3}+c_4z^{-4}$$

と展開表現した各係数(c_1，c_2，c_3，c_4)を求めよ．

解答2

式(52)より f_m を求める．すなわち，

$$f_1 = -p_1 = -10,\ f_2 = -\frac{1}{2}\times 30 = -15$$

$$f_3 = -\frac{1}{3}\times 100 = -\frac{100}{3},\ f_4 = -\frac{1}{4}\times 354 = -\frac{177}{2}$$

となり，さらに $h_m=f_m$ であるから，式(57)を利用する．途中の計算を含めて，以下に結果を示す．

- $m=1$ のとき

$$c_1=h_1c_0=-10$$

- $m=2$ のとき

$$2c_2=2h_2c_0+h_1c_1=70 \quad \rightarrow \quad c_2=35$$

- $m=3$ のとき

$$3c_3=3h_3c_0+2h_2c_1+h_1c_2=-150 \quad \rightarrow \quad c_3=-50$$

- $m=4$ のとき

$$4c_4=4h_4c_0+3h_3c_1+2h_2c_2+h_1c_3=96 \quad \rightarrow \quad c_4=24$$

よって，伝達関数は，

$$H(z)=1-10z^{-1}+35z^{-2}-50z^{-3}+24z^{-4}$$

と求められる．高校数学で学習した“根と係数の関係”を思い起こしてもらいたいもので，なかなかおもしろい計算方法である．

例題3

図15.13の設計仕様を満たす非再帰形ディジタルシステムの伝達関数をジョンソンの方法に基づき，設計せよ．

解答3

まず，**図15.13**の設計仕様より，振幅特性の対数を求める．

$$\ln(|G_0|) = \ln(1) = 0,\ \ln(|G_1|) = \ln(|G_7|) = \ln\left(\frac{1}{e}\right) = -1$$

$$\ln(|G_1|) = \ln\left(\frac{1}{e^2}\right) = -2 \;;\; \ell = 2,\ 3,\ 4,\ 5,\ 6$$

であり，位相を0とみなして，つまり，

$$G_\ell = |G_\ell|e^{j0} = |G_\ell| \quad ; \quad \ell = 0,\ 1,\ 2,\ \ldots,\ 7$$

の値を式(54)に当てはめて計算する(N=8)．計算結果を以下に示す．

$$h_0 = -1.5,\quad h_1 = h_7 = 0.42677,\ h_2 = h_6 = 0.25,$$
$$h_3 = h_5 = 0.07322,\quad h_4 = 0$$

次に，位相を付加してディジタルシステムを実現するわけだが，たとえば最小位相システムを考えてみる．詳細な説明は省略させてもらうが，最小位相ディジタルシステムの対数伝達関数の展開係数 $\left\{\hat{h}_m\right\}_{m=0}^{m=7}$ は，

$$\begin{cases} \hat{h}_0 = h_0 = -1.5, & \hat{h}_1 = 2h_1 = 0.85354, \\ \hat{h}_1 = 2h_2 = 0.5, & \hat{h}_3 = 2h_3 = 0.14644, \\ \hat{h}_4 = h_4 = 0, & \hat{h}_5 = 0, \\ \hat{h}_6 = 0, & \hat{h}_7 = 0 \end{cases}$$

で与えられる．この後のディジタルシステムの伝達関数の係数算出については，式(57)を利用して，例題2と同様の手順で計算することになる．たとえば，1次までの伝達関数近似であれば，式(57)より，

$$c_1=\hat{h}_1c_0=0.85354$$

となる．また，$h_0=-1.5$ より，式(50)に基づき，

$$H_0=e^{-1.5}=0.22313$$

であり，最終的に伝達関数 $H(z)$ は，

$$H(z)=0.22313(1+0.85354z^{-1})$$

で与えられる．一般的には，$\left\{\hat{h}_m\right\}_{m=0}^{m=7}$ に窓関数(2002年11月号，「FFTによる信号処理応用(データ処理II)」を参照)を掛けた値に基づき，ディジタルシステムの伝達関数を求めることが多い．

*　　　　*

今回で，DFT，FFTについての解説を終えることにし，次回からはJPEGやMPEGにおける画像圧縮技術としての基本的な信号処理(DCT，ウェーブレット変換など)を中心に解説していく予定である．お楽しみに．

みたに・まさあき　東京電機大学工学部情報通信工学科

音楽配信技術の最新動向

Vorbisfile library とAPIの概論

岸　哲夫

OggVorbisの話題を続けます．最新の情報として，OggVorbisに対応予定の携帯プレーヤの話題を紹介します．

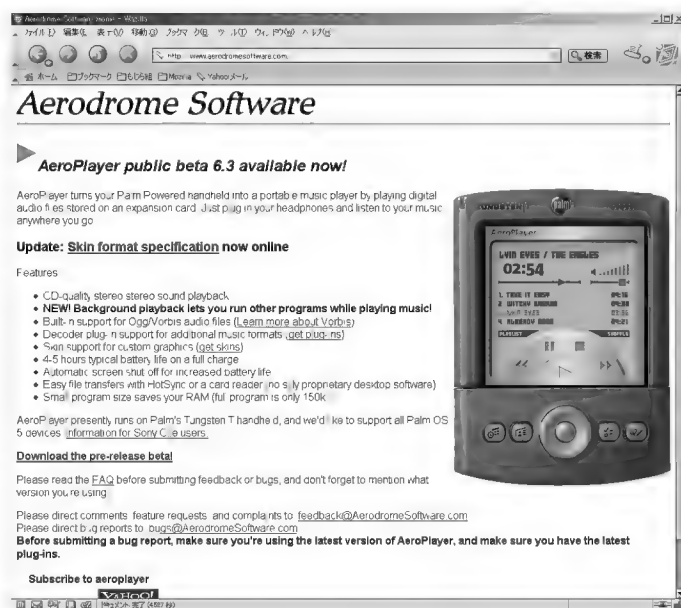
光SPDIF搭載のHD携帯プレーヤ「iHP100」が韓国のアイリバー社から製品化されるそうです．発売時にOggVorbisへの対応を行っているかどうかは疑問ですが，ほかのアイリバー社の製品と同様にファームウェアのバージョンアップで対応するようです．

この製品を使えば，光出力付きCDプレーヤを入力に使うことでコピーコントロールCDでもディジタイズできてしまいます．この仕様があたりまえになった場合，著作権問題がさらに複雑化してしまうことでしょう．

しかし画期的な製品であることはたしかです．アナログインターフェースも付いているので，録音にも使用できます．小さな筐体に10Gバイトまたは20GバイトのHDを内蔵し，発売時の対応音声形式はMP3，WMA，ASFです．USB2.0インターフェースも内蔵しています．

また，Linux対応ザウルスでVorbis対応のプレーヤが販売されていましたが，PalmOS5でVorbisの再生が行えるAeroPlayerのPublic Beta版が公開されました．ダウンロードは，次のURLから行えます．

```
http://www.aerodromesoftware.com/
```
（図1）

〔図1〕Aerodrome Software Home

〔図2〕Ogg Vorbis Documentation

Vorbisfile library

今回から数回に分けて，OggVorbisのプログラミングについて解説を行います．環境としてはLinuxを使用しますが，VC++を使ったWindows環境でも基本的な手順は同じです．

まずは，Vorbisfileについて解説します．

次に示すサイトにあるProgramming with vorbisfileという公式ドキュメントを元に解説を行います．

```
http://www.xiph.org/ogg/vorbis/docs.html
```
（図2）

● Vorbisfileライブラリについて

Vorbisfileライブラリを使用することで，Vorbisビットストリームの解読および基礎的な操作が可能になります．Cのソースで提供されているので，基本的にC以外の言語にリンクするこ

〔図3〕ライブラリの構成図

Libvorbisfile

libogg

Libvorbis

〔図4〕概念図

アプリケーション

OggVorbis Filestruct
形式構造体

入力データ
出力データ

API

〔図5〕vorbisdownload

とも可能です．いわゆるAPIを提供しています．

図3のように`Libvorbisfile`は，`libogg`および`libvorbis`ライブラリの上位層としてインプリメントされます．`Vorbisfile`を使用してデコードすることが，もっとも迅速かつ単純であると開発者は記述しています．

また，組み込み環境でのカスタマイズされたビットストリームI/Oルーチンを利用することもできると，ドキュメントには記述されています．

ただし，プログラム開発を行うのであれば「Programming with vorbisfile」を熟読することが必要だと思います．

Vorbisfile API概論

Vorbisfile libvorbisfileライブラリAPIの入出力インターフェースは，`OggVorbis_File`形式の構造体です．**図4**のように非常に単純な作りです．

大きく分けてVorbisfile APIは，次の機能的なカテゴリから成ります．

- 基本データ構造
- 初期化処理/後処理
- デコード処理
- 位置付け
- ファイルインフォメーション

上記の補足として，

- スレッド処理
- 標準入出力以外でのコールバックを使用した利用法について

Vorbisのライブラリは，下記のURLからダウンロード可能です．

`http://www.xiph.org/ogg/vorbis/download/`
（**図5**）

ここで`libao-0.8.3.tar.gz`，`libogg-1.0.tar.gz`，`libvorbis-1.0.tar.gz`，`vorbis-tools-1.0.tar.gz`をダウンロードすればよいでしょう．

なお，本項では，`libvorbis-1.0.tar.gz`を展開してください．構造体`OggVorbis_File`形式についてはAPIの項で説明します．

〔表1〕基本データ構造

データ形式	用　途
OggVorbis_File	この構造は基礎的なファイル情報を表す．それは物理的なファイルかビットストリーム，およびそのビットストリームに関するさまざまな情報へのポインタを含んでいる
vorbis_comment	この構造はファイルコメントを含んでいる．それは無制限のユーザーコメント，コメントの数に関する情報およびベンダ記述へのポインタを含んでいる
vorbis_info	この構造はビットストリームに関するエンコーダに関連する情報を含んでいる．それはエンコーダ情報，チャネル情報およびビットレート範囲を含んでいる
ov_callbacks	この構造は，ov_open_callbacks()によって使用されるファイル操作ルーチンへのポインタを含んでいる．「Using〔non stdio〕custom stream I/O via callbacks」の項を参照のこと

〔表3〕デコード処理

関　数	用　途
ov_read	この機能がデコード処理の肝である．すでに初期化された構造体OggVorbis_Fileを必要としている
ov_read_float	浮動小数点プロセッサを使用できるov_read

〔表4〕libvorbisfileをSeekingするAPI

関　数	用　途
ov_raw_seek	この関数はバイトで指定される．圧縮されたビットストリームの中の指定されたバイト位置へSeekする
ov_pcm_seek	この関数はPCMサンプルの中で指定される．特定のオーディオサンプル番号にSeekされる
ov_pcm_seek_page	この関数はPCMサンプルの中で指定され，指定されたオーディオサンプル番号に先行する最も近いページにSeekされる
ov_time_seek	この関数は秒の値で指定され，ビットストリームの中の特定の時間位置へSeekする
ov_time_seek_page	この関数は秒の値で指定され，ビットストリームの中の特定の時間位置に先行したもっとも近いページへSeekする

〔表2〕初期化処理/後処理

関　数	用　途
ov_open	Ogg Vorbisビットストリームを初期化する．ライブラリの他の機能を使用する前に，呼び出す
ov_open_callbacks	Ogg Vorbisビットストリームが使用する領域に加え，カスタムファイル/ビットストリーム操作ルーチンへアクセスするため領域を初期化する．標準入出力以外を使用する際にはこれを呼び出す
ov_test	対象のファイルがOgg Vorbisファイルかどうか不明な場合に使用する．成功の戻り値なら正しい形式．しかし，これだけでは構造体OggVorbis_Fileは実際のデコードのためにまだ完全には初期化されていない．正しい形式であることがわかって実際にデコードする際にはov_test_open()を完了する．なぜなら，対象のファイルはov_clear()を使用してクローズされたかもしれないからである．この呼び出しは完全なov_open()より負荷がかからない．注意：この関数から戻った後もlibvorbisfileがファイルリソースを所有しているのでfclose()しないようにする
ov_test_callbacks	上記の機能に加えてカスタムファイル/ビットストリーム操作ルーチンへアクセスするため領域を使用する際に，この関数を使う
ov_test_open	この関数呼び出しが成功後にov_test()またはov_test_callbacks()を使ってファイルをオープンする
ov_clear	ビットストリームを閉じて終了処理する．ビットストリームの使用を終えたときに呼ぶようにする．リターンの後には構造体OggVorbis_Fileは無効になる．再び使用する場合は初期化する

● 基本データ構造

内部で使用する基本データ構造は**表1**のとおりです．

ドキュメントに書いてあるように「vorbis/vorbisfile.h」および「vorbis/codec.h」のソースを見て解読してください．読んでわかる程度に理解していれば大丈夫だと思います．

詳しくは「公式ドキュメント」を参照してください．

● 初期化処理/後処理

ごく一般的な初期化処理と後処理のAPIを**表2**に示します．

標準入出力を使用したり，標準入出力以外でコールバックを使用した処理のための関数です．公式ドキュメントに簡単な手順が書いてあるので，それを引用します．

1）標準ライブラリのfopen()
2）ov_open()
3）主要な処理
4）ov_clear()

使用前には標準ライブラリのfopenが必要ですが，使用後にはov_clear()内でfcloseを実行しているとのことです．ov_clear()後にfcloseするとエラーになってしまうので注意してください．

● デコード処理

OggVorbisはPCMデータファイルを圧縮して符号化し，Ogg形式データを作ります．libvorbisfileをデコードするAPIはすべて，「vorbis/vorbisfile.h」の中で宣言されています．**表3**に関数を示します．

公式ドキュメントによると，デコード処理にもいくつかのパターンがあるということです．

1）multiple stream links
2）returned data amount
3）file cursor position

詳細は公式ドキュメントを参照してください．

● 位置付け

この機能は，デコードしはじめる位置を指定することを可能にします．libvorbisfileをSeekingするAPIはすべて，「vorbis/vorbisfile.h」の中で宣言されます（**表4**）．

通常はオーディオデータが始まる位置に位置付けする場合に使用します．そのほか，いろいろな状況またはファイルの状態によって位置付けすると効率が良い場合があります．

● ファイルインフォメーション

Libvorbisfileは，ビットストリーム属性に関する情報を得る多くの機能および解読するステータスを含んでいます．なお，libvorbisfileファイル情報ルーチンはすべて「vorbis/vorbisfile.h」の中で宣言されます(**表5**)．

● スレッド処理

スレッド処理についてある程度の知識があるならば，ぜひとも公式ドキュメントを読んでみてください．難しいことは書いてありません．スレッドを使用する場合のごく常識的なことが補足されています．

Vorbisfile APIは「スレッドセーフ」に作られているとのことです．プログラマはスレッドを扱う際の一般的な規則を守れば，問題なくスレッドに対応できるとのことです．細かい話については公式ドキュメントを熟読してください．

● 標準入出力以外でのコールバックを使用した利用法について

コールバックと標準入出力以外からのI/Oについて説明します．

stdioはどの環境にも存在し，便利で，何も考えずに使用できますが，すでにメモリ上に配置されているものを読み取りたい場合など，適合しない場合もあります．

そこでオリジナルなI/O機能を作って組み込むことが可能です．詳しくは公式ドキュメントを参照してください．

*　　　　　*

次回は，Vorbisfile APIについて詳細な解説を行います．

きし・てつお　オフィス岸

〔表5〕ファイル情報ルーチン

関　数	用　途
ov_bitrate	現在の論理的なビットストリームの平均値を返す
ov_bitrate_instant	最後にov_bitrateで返されたビットレート値を返す．最後に呼び出した値が存在しない場合に－1が戻る
ov_streams	指定した対象のビットストリームの論理的なビットストリームの数を返す
ov_seekable	ビットストリームをSeekできるかどうか判断する
ov_serialnumber	指定された論理的なビットストリームのユニークな通し番号を返す
ov_raw_total	物理的または論理的なSeek可能なビットストリーム中に含まれる圧縮されたバイトの合計値を返す
ov_pcm_total	物理的または論理的なSeek可能なビットストリーム中に含まれるサンプルの総数を返す
ov_time_total	物理的または論理的なSeek可能なビットストリーム中に含まれる時間の合計値を秒で返す
ov_raw_tell	ストリーム中で現在デコードが終了した場所の次のサンプルのバイト位置を返す
ov_pcm_tell	ストリーム中で現在デコードが終了した場所の次のサンプルの位置を返す
ov_time_tell	ストリーム中で現在デコードが終了した場所の次のサンプルの時間を秒で返す
ov_info	指定されたビットストリームセクションが使用する構造体vorbis_infoを返す
ov_comment	指定されたビットストリームが使用する構造体vorbis_commentを返す．Seekできないストリームについては，現在のビットストリームの構造体vorbis_commentを返す．返す際には指定のコメントを付加する

x86CPUだけでもマスタしたい

開発技術者のためのアセンブラ入門

第⑰回　論理，シフト，ローテート命令

大貫広幸

今回と次回で，x86系CPUがもっているビットやフラグの操作に関する命令について説明します．

ビット操作に関する命令としては，AND，OR，NOTといったビット単位の論理演算を行う「論理命令」と，指定ビット数分のシフトや回転を行う「シフト命令」や「ローテート命令」，そしてビットのテストやセット/リセットを行う「ビット命令」があります．

フラグ操作に関する命令としては，ステータスフラグの状態をバイト値として取得するための「バイト命令」，特定のフラグのセット/リセット，ロード/ストアを行う「フラグ制御命令」があります．

今回はこのうち，論理演算に関する命令とシフト，ローテートに関する命令について説明します．

論理演算

論理演算を行う命令としては，論理命令に分類されるAND，OR，XOR，NOTとビット命令に分類されているTESTの五つの命令があります(**表1**)．

ANDは論理積，ORは論理和，XORは排他的論理和，そしてNOTが論理否定の演算となります．TEST命令は，演算自体は論理積ですが，ビットパターンを調べるのが目的なので，演算結果の論理積の値は捨てられます．扱える値は，8ビット長のバイト整数，16ビット長のワード整数，そして32ビット長のダブルワード整数の3種類です．

● AND，OR，XOR命令

AND命令，OR命令，XOR命令は，演算が異なるのみでオペランドの指定は同じになっています．演算をop，オペランドの転送先をDEST，転送元をSOUで表すと，

DEST ← DEST op SOU

という演算をビットごとに行います(**表2**)．

転送先(DEST)には汎用レジスタやメモリ上の値が指定できます．そして，転送元(SOU)には汎用レジスタやメモリ上の値，イミディエイトが指定できます．ただし，メモリ上の値は，同時に二つ指定できないため，一つオペランドでメモリ上の値を指定した場合は，もう一方のオペランドは，汎用レジスタかイミディエイトとなります．そのため，論理演算のバリエーションは，

- 汎用レジスタ←汎用レジスタ op 汎用レジスタ
- 汎用レジスタ←汎用レジスタ op メモリ
- メモリ←メモリ op 汎用レジスタ
- 汎用レジスタ←汎用レジスタ op イミディエイト
- メモリ←メモリ op イミディエイト

となります．

論理演算実行後，ステータスフラグのOFとCFはゼロになり，SF，ZF，PFのフラグは演算結果にしたがい設定されます．そしてAFは

〔表1〕x86系の32ビットCPUで使用できる論理演算命令

分類	インストラクション名	動作	影響を受けるフラグ					
			OF	SF	ZF	AF	PF	CF
論理命令	AND	Logical AND ビット単位のAND(論理積)の演算 DEST ← DEST and SOU	0	*	*	?	*	0
	OR	Logical Inclusive OR ビット単位のOR(論理和)の演算 DEST ← DEST or SOU	0	*	*	?	*	0
	XOR	Logical Exclusive OR ビット単位のXOR(排他的論理和)の演算 DEST ← DEST xor SOU	0	*	*	?	*	0
	NOT	Logical NOT ビット単位のNOT(論理否定)の演算 DEST ← not DEST	·	·	·	·	·	·
ビット命令	TEST	Logical Compare ビット単位のAND(論理積)演算を行うが結果は捨てられる　SOU1 and SOU2	0	*	*	?	*	0

●表中のDESTはdestination(先)，SOU，SOU1，SOU2はsource(元)を表す
●表中の影響を受けるフラグの記号は次の状態を表す
　? = 未定義　· = 変化ない　* = 結果にしたがい変化する　0 = クリアされる

〔表2〕AND，OR，XOR，NOTの定義

元の値		演算結果			
		AND	OR	XOR	NOT
DEST	SOU	DEST	DEST	DEST	DEST
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

未定義の状態となります．実際のMASMでのAND命令の記述例を**リスト1**，gasでの記述例を**リスト2**に示します．

● TEST命令

TEST命令は，オペランドで指定された二つの整数値のANDの演算を行いますが，AND演算で得られた論理積の値は使用されず捨てられます．そのため，指定された二つのオペランドの値はともに転送元となり，変化しません．

このTEST命令の動作を，転送元1をSOU1，転送元2をSOU2で表すと，

SOU1 and SOU2

という論理演算をビットごとに行い，演算結果を捨てます．アセンブラの記述上，転送元1(SOU1)をsou1，転送元2(SOU2)をsou2とした場合，MASMでは，

TEST　sou1，sou2

と記述し，gasではsou1とsou2が逆になり，

TEST　sou2，sou1

と記述します．

転送元1(SOU1)には汎用レジスタやメモリ上の値が指定できます．そして，転送元2(SOU2)には汎用レジスタかイミディエイトが指定できます．そのため，TEST命令のバリエーションは，

〔リスト1〕MASMの論理命令とTEST命令の記述例

```
                               .586
                               .model flat
00000000                       .data
00000000 01                    dtByte  db   1
00000001 0002                  dtWord  dw   2
00000003 00000004              dtDWord dd   4
00000000                       .code
                               ;****************************************
                               ;    AND
                               ;****************************************
                               ; 汎用レジスタ←汎用レジスタ and 汎用レジスタ
00000000 22 F8                      and   bh,al
00000002 66| 23 DF                  and   bx,di
00000005 23 F9                      and   edi,ecx
                               ; 汎用レジスタ←汎用レジスタ and メモリ
00000007 22 1D 00000000 R           and   bl,dtByte
0000000D 66| 23 35                  and   si,dtWord
         00000001 R
00000014 23 0D 00000003 R           and   ecx,dtDWord
                               ; メモリ←メモリ and 汎用レジスタ
0000001A 20 35 00000000 R           and   dtByte,dh
00000020 66| 21 0D                  and   dtWord,cx
         00000001 R
00000027 21 1D 00000003 R           and   dtDWord,ebx
                               ; アキュムレータ←アキュムレータ and イミディエイト
0000002D 24 12                      and   al,12h
0000002F 66| 25 1234                and   ax,1234h
00000033 25 12345678                and   eax,12345678h
                               ; 汎用レジスタ←汎用レジスタ and イミディエイト
00000038 80 E7 12                   and   bh,12h
0000003B 66| 81 E1 1234             and   cx,1234h
00000040 81 E7 12345678             and   edi,12345678h
                               ; メモリ←メモリ and イミディエイト
00000046 80 25 00000000 R           and   dtByte,12h
         12
0000004D 66| 81 25                  and   dtWord,1234h
         00000001 R
         1234
00000056 81 25 00000003 R           and   dtDWord,12345678h
         12345678
                               ;****************************************
                               ;    TEST
                               ;****************************************
                               ; 汎用レジスタ and 汎用レジスタ
00000060 84 F8                      test  bh,al
00000062 66| 85 DF                  test  bx,di
00000065 85 F9                      test  edi,ecx
                               ; 汎用レジスタ and メモリ
00000067 84 1D 00000000 R           test  bl,dtByte
0000006D 66| 85 35                  test  si,dtWord
         00000001 R
00000074 85 0D 00000003 R           test  ecx,dtDWord
                               ; メモリ and 汎用レジスタ
0000007A 84 35 00000000 R           test  dtByte,dh
00000080 66| 85 0D                  test  dtWord,cx
         00000001 R
00000087 85 1D 00000003 R           test  dtDWord,ebx
                               ; アキュムレータ and イミディエイト
0000008D A8 12                      test  al,12h
0000008F 66| A9 1234                test  ax,1234h
00000093 A9 12345678                test  eax,12345678h
                               ; 汎用レジスタ and イミディエイト
00000098 F6 C7 12                   test  bh,12h
0000009B 66| F7 C1 1234             test  cx,1234h
000000A0 F7 C7 12345678             test  edi,12345678h
                               ; メモリ and イミディエイト
000000A6 F6 05 00000000 R           test  dtByte,12h
         12
000000AD 66| F7 05                  test  dtWord,1234h
         00000001 R
         1234
000000B6 F7 05 00000003 R           test  dtDWord,12345678h
         12345678
                               ;****************************************
                               ;    NOT
                               ;****************************************
000000C0 F6 D1                      not   cl
000000C2 66| F7 D1                  not   cx
000000C5 F7 D1                      not   ecx
000000C7 F6 15 00000000 R           not   dtByte
000000CD 66| F7 15                  not   dtWord
         00000001 R
000000D4 F7 15 00000003 R           not   dtDWord
                               end
```

- 汎用レジスタ and 汎用レジスタ
- メモリ and 汎用レジスタ
- 汎用レジスタ and イミディエイト
- メモリ and イミディエイト

となります.

これを見てわかるようにTEST命令には,

- 汎用レジスタ and メモリ

の機械語命令が存在しません. そのため, TEST命令では「汎用レジスタ and メモリ」の演算は,「メモリ and 汎用レジスタ」と置き換えて使用します.

ただし, MASMやgasなどの多くのx86系のアセンブラは,「汎用レジスタ and メモリ」のオペランドの指定を受け付けます. オペランドとして「汎用レジスタ and メモリ」が指定されていると, アセンブラは機械語命令の生成段階で「メモリ and 汎用レジスタ」とオペランドを置き換えてくれるため, MASMやgasなどのアセンブラを使用している分には「汎用レジスタ and メモリ」のオペランドも使用することができます.

TEST命令実行後のテータスフラグの変化は, AND命令と同じです. つまり, ステータスフラグのOFとCFはゼロになり, SF, ZF, PFのフラグは演算結果にしたがい設定されます. そしてAFは未定義の状態となります.

実際のMASMでのTEST命令の記述例を**リスト1**, gasでの記述例を**リスト2**に示します.

● NOT命令

NOT命令のオペランドは転送先一つで, オペランドで指定された転送先の汎用レジスタあるいはメモリ上の値をビットごとにNOT(論理否定)した値で置き換えます. つまり, オペランドの転送元をDESTで表すと,

〔リスト2〕gasの論理命令とTEST命令の記述例

```
 1                     .data
 2
 3  0000 01            dtByte:    .byte  1
 4  0001 0200          dtWord:    .word  2
 5  0003 04000000      dtDWord:   .long  4
 6
 7                     .text
 8                     #*************************************
 9                     #       AND
10                     #*************************************
11                     # 汎用レジスタ and 汎用レジスタ → 汎用レジスタ
12  0000 20C7                  andb    %al,%bh
13  0002 6621FB                andw    %di,%bx
14  0005 21CF                  andl    %ecx,%edi
15                     # 汎用レジスタ and メモリ → 汎用レジスタ
16  0007 221D0000              andb    dtByte,%bl
16       0000
17  000d 66233501              andw    dtWord,%si
17       000000
18  0014 230D0300              andl    dtDWord,%ecx
18       0000
19                     # メモリ and 汎用レジスタ → メモリ
20  001a 20350000              andb    %dh,dtByte
20       0000
21  0020 66210D01              andw    %cx,dtWord
21       000000
22  0027 211D0300              andl    %ebx,dtDWord
22       0000
23                     # アキュムレータ and イミディエイト → アキュムレータ
24  002d 2412                  andb    $0x12,%al
25  002f 66253412              andw    $0x1234,%ax
26  0033 25785634              andl    $0x12345678,%eax
26       12
27                     # 汎用レジスタ and イミディエイト → 汎用レジスタ
28  0038 80E712                andb    $0x12,%bh
29  003b 6681E134              andw    $0x1234,%cx
29       12
30  0040 81E77856              andl    $0x12345678,%edi
30       3412
31                     # メモリ and イミディエイト → メモリ
32  0046 80250000              andb    $0x12,dtByte
32     000012
33  004d 66812501              andw    $0x1234,dtWord
33       00000034
33       12
34  0056 81250300              andl    $0x12345678,dtDWord
34       00007856
34       3412
35
36
37
38
39
40
41
```

ここではAND{b, w, l}命令のみを示したが, OR{b, w, l}, XOR{b, w, l}命令も同じオペランドの指定で使用できる

```
42                     #*************************************
43                     #       TEST
44                     #*************************************
45                     # 汎用レジスタ and 汎用レジスタ
46  0060 84F8                  testb   %al,%bh
47  0062 6685DF                testw   %di,%bx
48  0065 85F9                  testl   %ecx,%edi
49                     # 汎用レジスタ and メモリ
50  0067 841D0000              testb   dtByte,%bl
50       0000
51  006d 66853501              testw   dtWord,%si
51       000000
52  0074 850D0300              testl   dtDWord,%ecx
52       0000
53                     # メモリ and 汎用レジスタ
54  007a 84350000              testb   %dh,dtByte
54       0000
55  0080 66850D01              testw   %cx,dtWord
55       000000
56  0087 851D0300              testl   %ebx,dtDWord
56       0000
57                     # アキュムレータ and イミディエイト
58  008d A812                  testb   $0x12,%al
59  008f 66A93412              testw   $0x1234,%ax
60  0093 A9785634              testl   $0x12345678,%eax
60       12
61                     # 汎用レジスタ and イミディエイト
62  0098 F6C712                testb   $0x12,%bh
63  009b 66F7C134              testw   $0x1234,%cx
63       12
64  00a0 F7C77856              testl   $0x12345678,%edi
64       3412
65                     # メモリ and イミディエイト
66  00a6 F6050000              testb   $0x12,dtByte
66       000012
67  00ad 66F70501              testw   $0x1234,dtWord
67       00000034
67       12
68  00b6 F7050300              testl   $0x12345678,dtDWord
68       00007856
68       3412
69                     #*************************************
70                     #       NOT
71                     #*************************************
72  00c0 F6D1                  notb    %cl
73  00c2 66F7D1                notw    %cx
74  00c5 F7D1                  notl    %ecx
75  00c7 F6150000              notb    dtByte
75       0000
76  00cd 66F71501              notw    dtWord
76       000000
77  00d4 F7150300              notl    dtDWord
77       0000
```

アセンブラgasも, 機械語にはない「汎用レジスタ and メモリ」の記述が使用できる

DEST ← not DEST

という演算をビットごとに行います(**表2**).

NOT命令は，実行してもステータスフラグ(OF, SF, ZF, AF, PF, CF)に影響を与えません．そのため，NOT命令を実行した後でもステータスフラグは変化しません．実際のMASMでのNOT命令の記述例を**リスト1**，gasでの記述例を**リスト2**に示します．

シフト命令とローテート命令

シフトもローテートも，レジスタやメモリ上の整数値を指定したビット数分だけ桁移動する命令です．

その違いは，シフト命令では移動時に補われるビットとしてゼロあるいは符号が使われ，移動により枠からはみ出したビットは捨てられます〔**図1(a)**〕．しかし，ローテート命令では，回転でビットが円を描くように移動します〔**図1(b)**〕．

シフト/ローテート命令で扱える値は，8ビット長のバイト整数，16ビット長のワード整数，そして32ビット長のダブルワード整数の3種類です．シフト命令には，SAR, SHR, SAL, SHL, SHRD, SHLDの6命令，ローテート命令にはROR, ROL, RCR, RCLの4命令があります(**表3**).

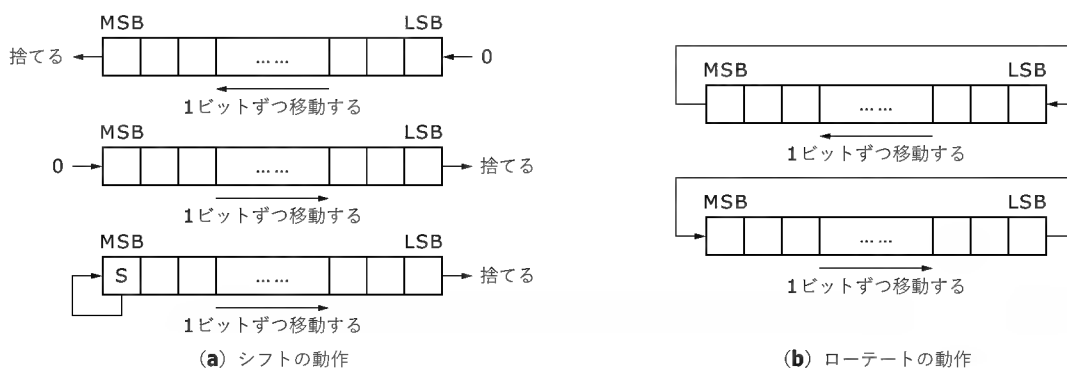
● SAR, SHR, SAL, SHL命令

この四つのシフト命令は，転送先とカウントという二つのオペランドを持ちます．

転送先には，汎用レジスタかメモリ上の値を指定します．シフト命令は，転送先で指定された値をリードし，シフトした値で転送先を置き換えます．

〔図1〕シフトとローテートの違い

(a) シフトの動作

(b) ローテートの動作

〔表3〕x86系の32ビットCPUで使用できるシフト/ローテート命令

分類	インストラクション名	動作	影響を受けるフラグ					
			OF	SF	ZF	AF	PF	CF
シフト命令	SAR	Shift Arithmetic Right(算術右シフト) 最上位ビット(MSB)の符号を変えずに 右に指定ビット数分シフトする	*	*	*	?	*	*
	SHR	Shift Logical Right(論理右シフト) 最上位ビット(MSB)にゼロが入るように 右に指定ビット数分シフトする	*	*	*	?	*	*
	SAL/SHL	Shift Arithmetic Left(算術左シフト) Shift Logical Left(論理左シフト) 最下位ビット(LSB)にゼロが入るように 左に指定ビット数分シフトする	*	*	*	?	*	*
	SHRD	Shift Right Double SOUとDESTを連続したビットと考え，右に指定 ビット数分シフトする．ただし，SOUは変化しない	?	*	*	?	*	*
	SHLD	Shift Left Double DESTとSOUを連続したビットと考え，左に指定 ビット数分シフトする．ただし，SOUは変化しない	?	*	*	?	*	*
ローテート命令	ROR	Rotate Right DESTのみを右に指定ビット数分回転する	*	・	・	・	・	*
	ROL	Rotate Left DESTのみを左に指定ビット数分回転する	*	・	・	・	・	*
	RCR	Rotate through Carry Right DESTとCFを右に指定ビット数分回転する	*	・	・	・	・	*
	RCL	Rotate through Carry Left DESTとCFを右に指定ビット数分回転する	*	・	・	・	・	*

表中のDESTはdestination(先)，SOUはsource(元)を表す
表中の影響を受けるフラグの記号は次の状態を表す
？＝未定義　・＝変化ない　＊＝結果にしたがい変化する

カウントには，イミディエイトあるいはレジスタ CL を指定します．また，カウント値は，イミディエイトもレジスタ CL も，8 ビットの符号なし整数で指定します．

ただし，実際にシフトで使われるビット数は，カウント値の下位 5 ビットから抽出された 0 ～ 31 の値で行われます．この四つのシフト命令の実際の記述例として，MASM を**リスト 3**，gas を**リスト 4** に示します．

(1) SAR 命令の動作

SAR 命令は，“ Shift Arithmetic Right ”の略で「算術右シフト」となります．符号付き 2 進整数を右にシフトするときに，この SAR 命令を使用します．実際の動作は，**図 2**(**a**)のように，転送先(DEST)を右に最上位ビット(MSB)の符号ビットを変化させずに，指定ビット数分シフトするものです．

(2) SHR 命令の動作

SHR 命令は，“ Shift Logical Right ”の略で「論理右シフト」となります．符号なし 2 進整数やビット列を右にシフトするときに，この SHR 命令を使用します．実際の動作は，**図 2**(**b**)のように，転送先(DEST)を右に最上位ビット(MSB)にゼロを補いながら，指定ビット数にシフトするものです．

(3) SAL，SHL 命令の動作

SAL は，“ Shift Arithmetic Left ”の略で「算術左シフト」，SHL は“ Shift Logical Left ”の略で「論理左シフト」となります．この SAL 命令と SHL 命令は，ニモニックが異なるのみで，動作も機械語命令もまったく同じものです(**リスト 3**，**リスト 4**)．

ニモニック名から，符号付き 2 進整数を左にシフトするときに SAL 命令を使用し，符号なし 2 進整数やビット列を左にシフトするときに SHL 命令を使用します．ただし，今述べたように SAL を使用しても SHL を使用しても，機械語命令自体は同一なので，動作も同じとなります．実際の動作は，**図 2**(**c**)のように，転送先(DEST)を左に最下位ビット(LSB)にゼロを補いながら，指定ビット数分シフトするものです．

(4) SAR，SHR，SAL，SHL 命令のフラグの変化

カウント(下位 5 ビット)がゼロの場合は，シフトは行われないため，ステータスフラグ(OF，SF，ZF，AF，PF，CF)は影響を受けません．カウントがゼロ以外ならシフトは実行されるため，ステータスフラグは次のような影響を受けます．

SF，ZF，PF は結果にしたがって設定され，AF は未定義となります．CF は最後にシフトされ，レジスタあるいはメモリ上の値から，外に出たビットを示します．ただし，SHL と SHR ではオペランドサイズ以上にシフトした場合のみ CF は未定義となります．

OF は，2 ビット以上シフトしたときは未定義，1 ビットシフトしたときは，次のようなシフトの影響を受けます．1 ビット左シフトの結果，CF と最上位ビット(MSB)が同じなら OF=0 となり，異なる場合は OF=1 となります．SAR での 1 ビット右シフトなら OF=0，SHR での 1 ビット右シフトなら OF は転送先の元の値の最上位ビットに設定されます．

● SHRD，SHLD 命令

二つの 16 ビットあるいは 32 ビット値を合わせたようなシフトを行います．SHRD が論理右シフト，SHLD が論理左シフトとなります．

オペランドは，転送先(DEST)と転送元(SOU)，カウント(COU)の三つのオペランドをもちます．そのため SHRD，SHLD 命令は，MASM では，

〔リスト 3〕**MASM のシフト命令の記述例**

```
                              .586
                              .model flat

00000000                       .data

00000000 01                    dtByte  db      1
00000001 0002                  dtWord  dw      2
00000003 00000004              dtDWord dd      4

00000000                       .code
00000000  D0 F8                        sar     al,1
00000002  66| D1 FA                    sar     dx,1
00000005  D1 FB                        sar     ebx,1
00000007  D0 3D 00000000 R             sar     dtByte,1
0000000D  66| D1 3D                    sar     dtWord,1
         00000001 R
00000014  D1 3D 00000003 R             sar     dtDWord,1
0000001A  D2 F8                        sar     al,cl
0000001C  66| D3 FA                    sar     dx,cl
0000001F  D3 FB                        sar     ebx,cl
00000021  D2 3D 00000000 R             sar     dtByte,cl
00000027  66| D3 3D                    sar     dtWord,cl
         00000001 R
0000002E  D3 3D 00000003 R             sar     dtDWord,cl
00000034  C0 F8 05                     sar     al,5
00000037  66| C1 FA 0A                 sar     dx,10
0000003B  C1 FB 15                     sar     ebx,21
0000003E  C0 3D 00000000 R             sar     dtByte,4
         04
00000045  66| C1 3D                    sar     dtWord,12
         00000001 R 0C
0000004D  C1 3D 00000003 R             sar     dtDWord,23
         17
00000054  D0 E0                        sal     al,1
00000056  D0 E0                        shl     al,1
00000058  66| D3 E2                    sal     dx,cl
0000005B  66| D3 E2                    shl     dx,cl
0000005E  C1 E3 15                     sal     ebx,21
00000061  C1 E3 15                     shl     ebx,21
00000064  D0 25 00000000 R             sal     dtByte,1
0000006A  D0 25 00000000 R             shl     dtByte,1
00000070  66| D3 25                    sal     dtWord,cl
         00000001 R
00000077  66| D3 25                    shl     dtWord,cl
         00000001 R
0000007E  C1 25 00000003 R             sal     dtDWord,23
         17
00000085  C1 25 00000003 R             shl     dtDWord,23
         17
                                       end
```

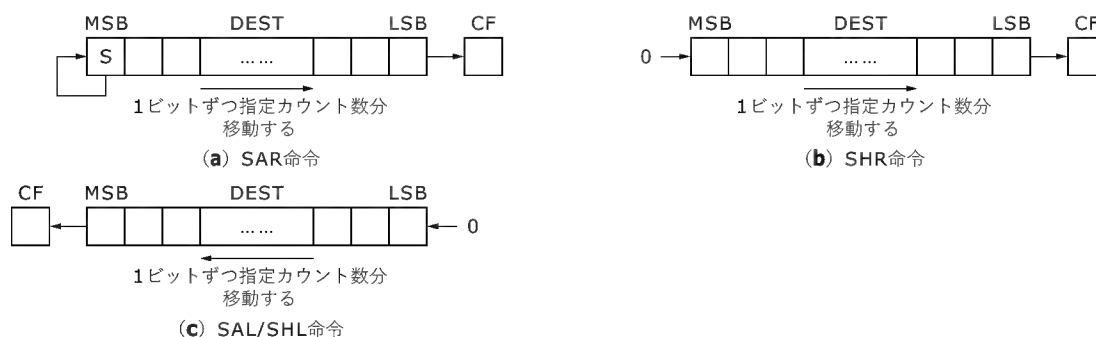
〔リスト4〕gasのシフト命令の記述例

```
1                        .data
2
3   0000 01              dtByte:    .byte  1
4   0001 0200            dtWord:    .word  2
5   0003 04000000        dtDWord:   .long  4
6
7                        .text
8   0000 D0F8                      sarb    %al
9   0002 66D1FA                    sarw    %dx
10  0005 D1FB                      sarl    %ebx
11  0007 D03D0000                  sarb    dtByte
11       0000
12  000d 66D13D01                  sarw    dtWord
12       000000
13  0014 D13D0300                  sarl    dtDWord
13       0000
14  001a D2F8                      sarb    %cl,%al
15  001c 66D3FA                    sarw    %cl,%dx
16  001f D3FB                      sarl    %cl,%ebx
17  0021 D23D0000                  sarb    %cl,dtByte
17       0000
18  0027 66D33D01                  sarw    %cl,dtWord
18       000000
19  002e D33D0300                  sarl    %cl,dtDWord
19       0000
20  0034 C0F805                    sarb    $5,%al
21  0037 66C1FA0A                  sarw    $10,%dx
22  003b C1FB15                    sarl    $21,%ebx
23  003e C03D0000                  sarb    $4,dtByte
23       000004
24  0045 66C13D01                  sarw    $12,dtWord
24       0000000C
25  004d C13D0300                  sarl    $23,dtDWord
25       000017
26
27
28
29
30
31
32
33  0054 D0E0                      salb    %al
34  0056 D0E0                      shlb    %al
35  0058 66D3E2                    salw    %cl,%dx
36  005b 66D3E2                    shlw    %cl,%dx
37  005e C1E315                    sall    $21,%ebx
38  0061 C1E315                    shll    $21,%ebx
39  0064 D0250000                  salb    dtByte
39       0000
40  006a D0250000                  shlb    dtByte
40       0000
41  0070 66D32501                  salw    %cl,dtWord
41       000000
42  0077 66D32501                  shlw    %cl,dtWord
42       000000
43  007e C1250300                  sall    $23,dtDWord
43       000017
44  0085 C1250300                  shll    $23,dtDWord
44       000017
```


〔図2〕SAR，SHR，SAL，SHL命令の動作

(a) SAR命令

(b) SHR命令

(c) SAL/SHL命令

　　SHRD　　dest，sou，cou

　　SHLD　　dest，sou，cou

と記述しますが，gasでは，

　　SHRD　　cou，sou，dest

　　SHLD　　cou，sou，dest

と記述することになります．

転送先(DEST)には，16ビットあるいは32ビットの汎用レジスタかメモリ上の値を指定します．転送元(SOU)には，転送先(DEST)と同じサイズの汎用レジスタ，カウント(COU)にはイミディエイトかレジスタCLを指定します．

カウント値は，イミディエイトもレジスタCLも，8ビットの符号なし整数で指定しますが，実際にシフトに使用するビット数は，カウント値の下位5ビットから抽出した0～31の値で行われます．

SHRD命令の動作は，まず転送元(SOU)を上位，転送先(DEST)を下位とするような32ビットあるいは64ビット長の一時的な値を作ります．

この一時的な値を指定カウント値(下位5ビット)のビット数分，右にシフトします．

そして最後に，転送先(DEST)のオペランドサイズ分，シフト結果の下位ビットを抽出し，転送先(DEST)に転送します〔**図3(a)**〕

SHLD命令の動作は，まず転送先(DEST)を上位，転送元(SOU)を下位とするような32ビットあるいは64ビット長の一時的な値を作ります．この一時的な値を指定カウント値(下位5ビット)のビット数分，左にシフトします．

そして最後に，転送先(DEST)のオペランドサイズ分，シフト結果の上位ビットを抽出し，転送先(DEST)に転送します〔**図3(b)**〕．

カウント(下位5ビット)がゼロの場合は，シフトは行われな

いため，ステータスフラグ(OF，SF，ZF，AF，PF，CF)は影響を受けません．カウントがゼロ以外ならシフトは実行されるため，ステータスフラグは次のような影響を受けます．

SF，ZF，PFは結果にしたがって設定され，AFは未定義となります．CFは最後にシフトされ，32ビットあるいは64ビット長の一時的な値の外に出たビットを示します．

OFは，2ビット以上シフトしたときは未定義，1ビットシフトしたときは，シフトの結果，符号が変化したときはOF=1，符号が変化しなかった場合はOF=0となります．

オペランドサイズより多いシフトをした場合は転送先(DEST)とフラグは未定義となります．

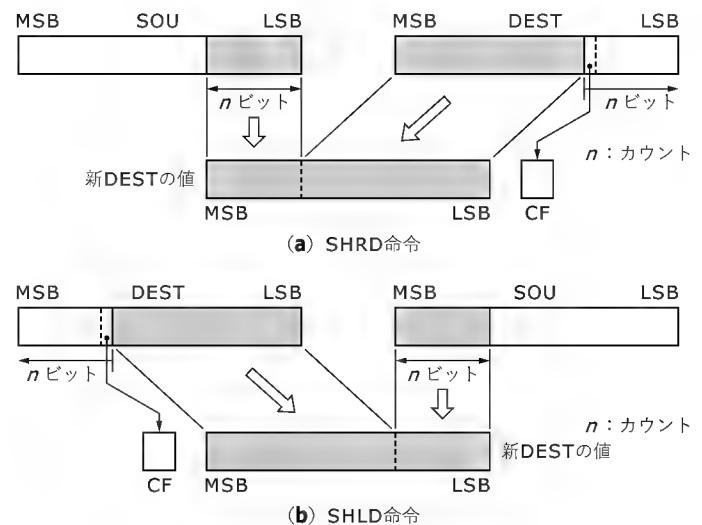
実際のMASMでの記述例を**リスト5**，gasでの記述例を**リスト6**に示します．

● ROR，ROL，RCR，RCL命令

この四つのローテート命令は，転送先とカウントという二つのオペランドを持ちます．

転送先には，汎用レジスタかメモリ上の値を指定します．ローテート命令は，転送先で指定された値をリードし，回転した値で転送先を置き換えます．

カウントには，イミディエイトあるいはレジスタCLを指定します．また，カウント値は，イミディエイトもレジスタCLも，8ビットの符号なし整数で指定します．

ただし，シフトする実際のビット数は，カウント値の下位5ビットから抽出された0～31の値で行われます．

ローテート命令の実際のMASMでの記述例を**リスト7**，gasでの記述例を**リスト8**に示します．

(1) ROR，ROL命令の動作

RORは右回転，ROLは左回転のローテートを行います．実際の動作は，**図4(a)**と**図4(b)**のように転送先内のビットのみが指定カウント値(下位5ビット)分，回転します．

〔図3〕SHRD，SHLD命令の動作

(a) SHRD命令

(b) SHLD命令

〔リスト5〕MASMのSHRD，SHLD命令の記述例

```
                                .586
                                .model flat

00000000                        .data

00000000 01                     dtByte  db      1 ─転送先(DEST)
00000001 0002                   dtWord  dw      2 ─転送元(SOU)
00000003 00000004               dtDWord dd      4  カウント(COU)

00000000                        .code
00000000  66| 0F AC D0                  shrd    ax,dx,7
          07
00000005  66| 0F AD DE                  shrd    si,bx,cl
00000009  66| 0F AC 05                  shrd    dtWord,ax,3
          00000001 R 03
00000012  66| 0F AD 3D                  shrd    dtWord,di,cl
          00000001 R
0000001A  0F AC D0 08                   shrd    eax,edx,8
0000001E  0F AD DE                      shrd    esi,ebx,cl
00000021  0F AC 05                      shrd    dtDWord,eax,13
          00000003 R 0D
00000029  0F AD 3D                      shrd    dtDWord,edi,cl
          00000003 R
                                ;
00000030  66| 0F A4 D0                  shld    ax,dx,7
          07
00000035  66| 0F A5 DE                  shld    si,bx,cl
00000039  66| 0F A4 05                  shld    dtWord,ax,3
          00000001 R 03
00000042  66| 0F A5 3D                  shld    dtWord,di,cl
          00000001 R
0000004A  0F A4 D0 08                   shld    eax,edx,8
0000004E  0F A5 DE                      shld    esi,ebx,cl
00000051  0F A4 05                      shld    dtDWord,eax,13
          00000003 R 0D
00000059  0F A5 3D                      shld    dtDWord,edi,cl
           00000003 R

                                 end
```

〔リスト6〕gasのSHRD，SHLD命令の記述例

```
 1                      .data
 2
 3  0000 01             dtByte:   .byte  1 ─カウント(COU)
 4  0001 0200           dtWord:   .word  2 ─転送元(SOU)
 5  0003 04000000       dtDWord:  .long  4  転送先(DEST)
 6
 7                      .text
 8  0000 660FACD0               shrdw   $7,%dx,%ax
 8       07
 9  0005 660FADDE               shrdw   %cl,%bx,%si
10  0009 660FAC05               shrdw   $3,%ax,dtWord
10       01000000
10       03
11  0012 660FAD3D               shrdw   %cl,%di,dtWord
11       01000000
12  001a 0FACD008               shrdl   $8,%edx,%eax
13  001e 0FADDE                 shrdl   %cl,%ebx,%esi
14  0021 0FAC0503               shrdl   $13,%eax,dtDWord
14       0000000D
15  0029 0FAD3D03               shrdl   %cl,%edi,dtDWord
15       000000
16                      #
17  0030 660FA4D0               shldw   $7,%dx,%ax
17       07
18  0035 660FA5DE               shldw   %cl,%bx,%si
19  0039 660FA405               shldw   $3,%ax,dtWord
19       01000000
19       03
20  0042 660FA53D               shldw   %cl,%di,dtWord
20       01000000
21  004a 0FA4D008               shldl   $8,%edx,%eax
22  004e 0FA5DE                 shldl   %cl,%ebx,%esi
23  0051 0FA40503               shldl   $13,%eax,dtDWord
23       0000000D
24  0059 0FA53D03               shldl   %cl,%edi,dtDWord
24       000000
25
```

〔リスト7〕MASMのローテート命令の記述例

```
                              .586
                              .model flat

00000000                      .data

00000000 01                   dtByte  db      1
00000001 0002                 dtWord  dw      2
00000003 00000004             dtDWord dd      4

00000000                      .code
00000000  D0 C8                       ror     al,1
00000002  66| D1 CA                   ror     dx,1
00000005  D1 CB                       ror     ebx,1
00000007  D0 0D 00000000 R            ror     dtByte,1
0000000D  66| D1 0D                   ror     dtWord,1
          00000001 R
00000014  D1 0D 00000003 R            ror     dtDWord,1
0000001A  D2 C8                       ror     al,cl
0000001C  66| D3 CA                   ror     dx,cl
0000001F  D3 CB                       ror     ebx,cl
00000021  D2 0D 00000000 R            ror     dtByte,cl
00000027  66| D3 0D                   ror     dtWord,cl
          00000001 R
0000002E  D3 0D 00000003 R            ror     dtDWord,cl
00000034  C0 C8 05                    ror     al,5
00000037  66| C1 CA 0A                ror     dx,10
0000003B  C1 CB 15                    ror     ebx,21
0000003E  C0 0D 00000000 R            ror     dtByte,4
          04
00000045  66| C1 0D                   ror     dtWord,12
          00000001 R 0C
0000004D  C1 0D 00000003 R            ror     dtDWord,23
          17
                               end
```

ここではROR命令のみを示したが，ROL，RCR，RCLの命令も同じオペランドの指定で使用できる

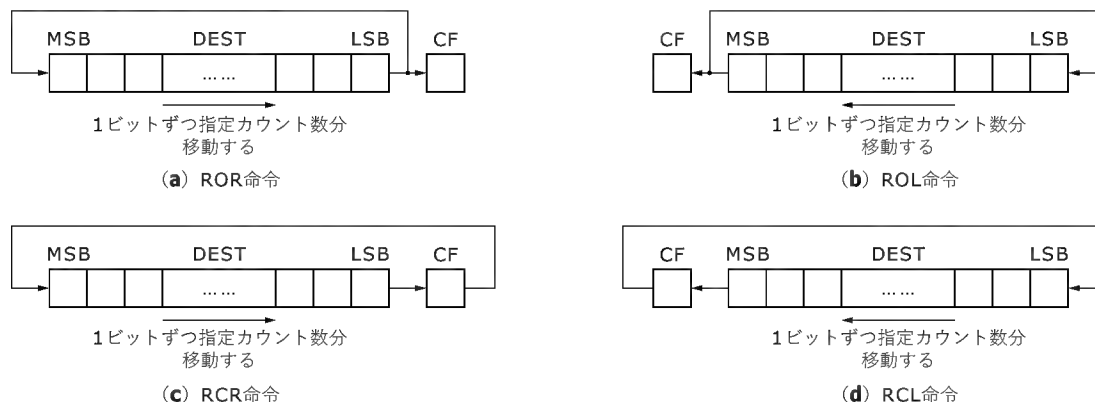
〔リスト8〕gasのローテート命令の記述例

```
1                      .data
2
3   0000 01            dtByte:   .byte  1
4   0001 0200          dtWord:   .word  2
5   0003 04000000      dtDWord:  .long  4
6
7                      .text
8   0000 D0C8                 rorb    %al
9   0002 66D1CA               rorw    %dx
10  0005 D1CB                 rorl    %ebx
11  0007 D00D0000             rorb    dtByte
11       0000
12  000d 66D10D01             rorw    dtWord
12       000000
13  0014 D10D0300             rorl    dtDWord
13       0000
14  001a D2C8                 rorb    %cl,%al
15  001c 66D3CA               rorw    %cl,%dx
16  001f D3CB                 rorl    %cl,%ebx
17  0021 D20D0000             rorb    %cl,dtByte
17       0000
18  0027 66D30D01             rorw    %cl,dtWord
18       000000
19  002e D30D0300             rorl    %cl,dtDWord
19       0000
20  0034 C0C805               rorb    $5,%al
21  0037 66C1CA0A             rorw    $10,%dx
22  003b C1CB15               rorl    $21,%ebx
23  003e C00D0000             rorb    $4,dtByte
23       000004
24  0045 66C10D01             rorw    $12,dtWord
24       0000000C
25  004d C10D0300             rorl    $23,dtDWord
25       000017
```


〔図4〕ROR，ROL，RCR，RCLの動作

(a) ROR命令

(b) ROL命令

(c) RCR命令

(d) RCL命令

(2) RCR，RCL命令の動作

RCRは右回転，RCLは左回転のローテートを行います．実際の動作は，**図4(c)**と**図4(d)**のように転送先とCFのビットが指定カウント値(下位5ビット)分，回転します．

(3) ROR，ROL，RCR，RCL命令のフラグの変化

ローテート命令では，SF，ZF，AF，PFは影響を受けません．CFは，ローテートの結果，そこに移動したビットの値を示します．OFは，2ビット以上ローテートしたときは未定義，1ビットローテートしたときは影響を受けます．1ビット左ローテートの場合，結果のCFと最上位ビット(MSB)をXORした値がOFに設定されます．1ビット右ローテートの場合は，結果の上位2ビットをXORした値がOFに設定されます．

*　　　　*

次回は，ビットやフラグの操作に関する命令のうち，今回説明しなかったビット/バイト命令とフラグ制御命令について説明する予定です．

おおぬき・ひろゆき　大貫ソフトウェア設計事務所

第8回 C言語における GCCの拡張機能(3)

岸　哲夫

今回は，前回に引き続きGNU Cで使用できる拡張機能について説明と検証を行う．拡張機能には，使用することにより効率的なコードを生成できるもの，簡潔な表現が行えるものなども多い．しかし，それらを使用することによる可搬性の低下なども起こりうる．これらについて，実際にコンパイルを行った結果を見ながら，解説を行う．（編集部）

前回に続いてGNU Cの拡張機能について詳細に説明と検証を行います．

● プロトタイプ宣言と古い方式の関数定義

通常，GNU Cでコードを書く場合に，関数定義のプロトタイプ宣言を省略する人はいないと思います．省略した場合には以下のような問題が発生します．

●使用する関数は使用される前に定義されていなければならない

関数が数十数百ある場合にそんなことを考えるのは不毛です．

●関数に渡す引き数が正しくなくてもコンパイルが通ってしまう

一見矛盾なく動いてしまいますが，後々おかしなことになります．

●戻り値の型を間違えて戻してもエラーにならない

charだと思っていた戻り値が，じつはlongで，しかもその値を別の関数の呼び出しに使って落ちた……などという場合には，かなり見つけにくいバグになるでしょう．

連載第5回（本誌2003年1月号）で詳細に記してありますが，もしプロトタイプ宣言をしていない古いソースを使わなければならない場合は，protoizeコマンドを使用するなどして関数定義のプロトタイプ宣言を付加するべきだと思います．

リスト1～リスト4の例では，関数の型の暗黙の型宣言がなされてしまい，結果としてまったく違う値を返してしまっています．

まずプロトタイプを使用しない場合の結果は，次のようになります．

```
$ gcc    -o test83 test83.c
test83.c:13: warning: type mismatch with
         previous implicit declaration
test83.c:7: warning: previous implicit
         declaration of `Double2'
test83.c:13: warning: `Double2' was previously
         implicitly declared to return `int'
$ ./test83
start
6.000000
$
```

プロトタイプを使用した結果は，次のようになります．

```
$ gcc    -o test84 test84.c
$ ./test84
start
4.000000
$
```

この例では意図的にエラーとなる場合を挙げていますが，実際にこのようなことが数多く起こりえます．

test83ではプロトタイプを使用しないため，関数の型に対し暗黙の型変換がなされてしまい，double Double2(double result)という関数がint Double2(int result)とコンパ

〔リスト1〕プロトタイプを使わない例(test83.c)

```
#include <stdio.h>
//double Double2(double result);
int main(void)
{
    double result;
    printf("start¥n");
    result = Double2(2);
    printf("%lf¥n",result);
    return 0;
}

double Double2(double result)
{
    return result * result;
}
```

〔リスト2〕プロトタイプを使った例(test84.c)

```
#include <stdio.h>
double Double2(double result);
int main(void)
{
    double result;
    printf("start¥n");
    result = Double2(2);
    printf("%lf¥n",result);
    return 0;
}

double Double2(double result)
{
    return result * result;
}
```

〔リスト3〕プロトタイプを使わない例から生成されたアセンブラ（test83.s）

```
	.file	"test83.c"
	.version	"01.01"
gcc2 compiled.:
.section	.rodata
.LC0:
	.string "start¥n"
.LC1:
	.string "%lf¥n"
.text
	.align 4
.globl main
	.type	main,@function
main:
	pushl	%ebp
	movl	%esp,%ebp
	subl	$24,%esp
	addl	$-12,%esp
	pushl	$.LC0
	call	printf
	addl	$16,%esp
	addl	$-12,%esp
	pushl	$2
	call	Double2
	addl	$16,%esp
	movl	%eax,%eax
	movl	%eax,-12(%ebp)
	fildl	-12(%ebp)
	fstpl	-8(%ebp)
	addl	$-4,%esp
	fldl	-8(%ebp)
	subl	$8,%esp
	fstpl	(%esp)
	pushl	$.LC1
	call	printf
	addl	$16,%esp
	xorl	%eax,%eax
	jmp .L2
	.p2align	4,,7
.L2:
	movl	%ebp,%esp
	popl	%ebp
	ret
.Lfe1:
	.size	main,.Lfe1-main
	.align 4
.globl Double2
	.type	Double2,@function
Double2:
	pushl	%ebp
	movl	%esp,%ebp
	fldl	8(%ebp)
	fmull	8(%ebp)
	jmp .L3
	.p2align	4,,7
.L3:
	movl	%ebp,%esp
	popl	%ebp
	ret
.Lfe2:
	.size	Double2,.Lfe2-Double2
	.ident	"GCC: (GNU) 2.95.3 20010315 (release)"
```

〔リスト4〕プロトタイプを使った例から生成されたアセンブラ（test84.s）

```
	.file	"test84.c"
	.version	"01.01"
gcc2 compiled.:
.section	.rodata
.LC0:
	.string "start¥n"
.LC2:
	.string "%lf¥n"
	.align 8
.LC1:
	.long	0x0,0x40000000
.text
	.align 4
.globl main
	.type	main,@function
main:
	pushl	%ebp
	movl	%esp,%ebp
	subl	$24,%esp
	addl	$-12,%esp
	pushl	$.LC0
	call	printf
	addl	$16,%esp
	addl	$-8,%esp
	fldl	.LC1
	subl	$8,%esp
	fstpl	(%esp)
	call	Double2
	addl	$16,%esp
	fstpl	-8(%ebp)
	addl	$-4,%esp
	fldl	-8(%ebp)
	subl	$8,%esp
	fstpl	(%esp)
	pushl	$.LC2
	call	printf
	addl	$16,%esp
	xorl	%eax,%eax
	jmp .L2
	.p2align	4,,7
.L2:
	movl	%ebp,%esp
	popl	%ebp
	ret
.Lfe1:
	.size	main,.Lfe1-main
	.align 4
.globl Double2
	.type	Double2,@function
Double2:
	pushl	%ebp
	movl	%esp,%ebp
	fldl	8(%ebp)
	fmull	8(%ebp)
	jmp .L3
	.p2align	4,,7
.L3:
	movl	%ebp,%esp
	popl	%ebp
	ret
.Lfe2:
	.size	Double2,.Lfe2-Double2
	.ident	"GCC: (GNU) 2.95.3 20010315 (release)"
```

〔リスト5〕関数定義が古いままのソース（test85.c）

```
#include <stdio.h>
typedef short	t_id;
int foo(t_id x);
int main(void)
{
	int result;
	t_id y;
	printf("start¥n");
	y	=	2;
	result = foo(y);
	printf("%d¥n",result);
	return 0;
}

int foo(x)
	t_id x;
{
	return x * x;
}
```

〔リスト6〕関数定義を修正したソース（test87.c）

```
#include <stdio.h>
typedef short	t_id;
int foo(t_id x);
int main(void)
{
	int result;
	t_id y;
	printf("start¥n");
	y	=	2;
	result = foo(y);
	printf("%d¥n",result);
	return 0;
}

int foo(t_id x)
{
	return x * x;
}
```

イラに解釈されてしまったのです．アセンブラを見るとわかるように引き数も，戻り値も間違っています．

プロトタイプの前置きが長くなりましたが，本題です．

古いソースにprotoizeを使用してプロトタイプを付加したとしても関数の定義が古いままだと，ANSI Cの場合，不具合が発生することがあります．GNU Cの拡張機能として**リスト5**と**リスト6**の二つのコードが同等になります．

拡張機能を使わない方法でtest85.cをコンパイルした結果は，次のようになります．

```
$ gcc  -pedantic test85.c -o test85
test85.c:1: warning: carriage return in
         preprocessing directive
test85.c:2: warning: carriage return in
         source file
test85.c:2: warning: (we only warn about
         the first carriage return)
test85.c: In function `foo':
test85.c:17: warning: promoted argument `x'
         doesn't match prototype
test85.c:3: warning: prototype declaration
$
```

拡張機能を使った場合，次のようになります．

```
$ gcc  test85.c -o test85
$
```

〔リスト7〕C++方式のコメントを使用したソース(test88.c)

```
#include <stdio.h>
//コメント
typedef short   t id;
int foo(t id x);
int main(void)
{
    int result;
    t id y;
    printf("start¥n");
    y   =   2;
    result = foo(y);
    printf("%d¥n",result);
    return 0;
}

int foo(t id x)
{
    return x * x;
}
```

拡張機能を使わない場合，関数fooの引き数の型が暗黙にintとなっています．プロトタイプ宣言で指定した型が間違っているとワーニングメッセージが出てしまいます．

拡張機能を使用した場合，正しく認識されています．つまり，プロトタイプ宣言のほうが正しいとみなしてコンパイルをしています．

この機能を使用する場合は，ANSIでコンパイルしないようにコメントなどを記述し，明示すべきです．便利な機能ですが，可読性に問題が発生するでしょう．

● コメントの形式

C++ではコメントを次のように記述します．

```
#include <stdio.h>
//typedef  short  t_id;  //コメント行
int foo(t_id x);
int main(void)
```

もちろんオプションに-ansiまたは-traditionalを指定した場合には，C++方式のコメントは認識されません．ほかの多くのCの実装でもこのようなコメントを使うことができるので，ソースをANSIに限定しなくてよいのなら，このコメント形式を使って問題ないでしょう(**リスト7**)．

-ansiオプションでコンパイルした場合，次のようにエラーとなります．

```
$ gcc  -S -ansi  test88.c
test88.c:2: parse error before `/'
test88.c:4: parse error before `x'
test88.c: In function `main':
test88.c:8: `t_id' undeclared (first use in
          this function)
test88.c:8: (Each undeclared identifier is
          reported only once
test88.c:8: for each function it appears in.)
test88.c:8: parse error before `y'
test88.c:10: `y' undeclared (first use in
          this function)
test88.c: At top level:
test88.c:16: parse error before `x'
test88.c: In function `foo':
test88.c:18: `x' undeclared (first use in
          this function)
$
```

〔リスト8〕識別子の名前にドル記号を使用したソース(test89.c)

```
#include <stdio.h>
typedef short   t_id;
int foo(t id x);
int main(void)
{
    int $result;
    t id y;
    printf("start¥n");
    y   =   2;
    $result = foo(y);
    printf("%d¥n",$result);
    return 0;
}

int foo(t id x)
{
    return x * x;
}
```

● 識別子の名前の中のドル記号

GNU Cでは識別子の名前の中でドル記号($)を使うことができます．もちろん古いCコンパイラでもそのような識別子を使うことを許しているものもあります．しかし，識別子の中のドル記号がサポートされないターゲットマシンもあります．その理由として，ターゲットマシンのアセンブラが識別子の中のドル記号を許さないことがあるからです．

そのようなターゲットマシン用にクロスコンパイルをする場合には，オプション-pedanticをつけてコンパイルするとエラーになるのでわかりやすいでしょう(**リスト8**)．

-pedanticオプションでコンパイルした場合，次のようにエラーとなります．なお連載第3回(本誌2002年10月号)の「警告を要求/抑止するオプション」でpedanticオプションについて説明しています．

```
$ gcc  -S -pedantic  test89.c
test89.c:6: warning: `$' in identifier
test89.c:10: warning: `$' in identifier
test89.c:11: warning: `$' in identifier
test89.c: In function `main':
test89.c:6: warning: `$' in identifier
test89.c:10: warning: `$' in identifier
test89.c:11: warning: `$' in identifier
$
```

● ESC文字について

メールの文字列中などで，[ESC]文字と$Bや(Bなどの文字を組み合わせた「エスケープシーケンス」で，文字セットを切り替えています．JIS-1983規格に切り替える際には[ESC] $ Bを文字列中に埋め込みます．

そのような用途に使う場合に，文字列リテラルとして“\e$B”のように定義ができます(**リスト9**)．

```
$ gcc  -S -pedantic  test90.c
test90.c: In function `main':
test90.c:4: warning: non-ANSI-standard
          escape sequence, `¥e'
test90.c:5: warning: non-ANSI-standard
          escape sequence, `¥e'
$
```

拡張機能を使用した場合にワーニングを出す指定をすると，

〔リスト 9〕**ESC文字を使用したソース**(test90.c)

```
#include <stdio.h>
int main(void)
{
    printf("¥e$B");
    printf("¥e$(0");
    return 0;
}
```

上記のようになります.

● 型あるいは変数のアラインメントを問い合わせる

たとえば,sizeofを使う場合と同様に__alignof__で,あるオブジェクトがどのようにアラインメントされるかを問い合わせることができます.実行環境によっては,アラインメントを必要としないものもあります.そのような場合に__alignof__は型の推奨アラインメントを報告します.

__alignof__のオペランドがlvalueの場合には,__alignof__の値はそのlvalueが取るとわかっているアラインメントの値のうち最大の値となります.

このアラインメントの値は,そのlvalueのデータ型から決められることもありますし,そのlvalueが構造体の一部である場合には,その構造体からアラインメントの値を継承することもあります(**リスト 10**,**リスト 11**).

上記のソースの実行結果は,次のようになります.

```
$ gcc  -o test91 test91.c
$ ./test91
doubleの境界----8
floatの境界----4
long longの境界----8
intの境界----4
charの境界----1
fooの境界----8
foo.eの境界----1
$
```

おそらく身近にあるインテルまたはインテル互換ではない環境,たとえばMac OS X上のGCCでは,また違った結果になると思います.

ちなみに第5回連載で作成したSHプロセッサ向けの環境でコンパイルすると,次のようになります(**リスト 12**).

```
$ sh-hitachi-coff-gcc -O3 -S test91.c
```

上記のようにchar,foo.eは1バイト,その他は4バイト境界になっています.

● 変数の属性の指定

キーワード__attribute__により,変数または構造体フィールドに特別な属性を指定することができます.このキーワードの後に,二重の丸括弧(())に囲まれた属性指定が続きます.現在,8個の属性aligned,mode,nocommon,packed,section,transparent_union,unused,weakが変数に対して定義されています.その他の属性が,前述の関数および後述する型に対して定義されています.

〔リスト 10〕**型あるいはアラインメントを問い合わせたソース**(test91.c)

```
#include <stdio.h>
//型あるいは変数のアラインメントを問い合わせる

int main(void)
{
    double  a;
    float   b;
    long long   c;
    int d;
    char    e;
    struct foo {
        int d;
        char    e;
    } foo1;
    printf("doubleの境界----%d¥n", __alignof__(a));
    printf("floatの境界----%d¥n", __alignof__(b));
    printf("long longの境界----%d¥n", __alignof__(c));
    printf("intの境界----%d¥n", __alignof__(d));
    printf("charの境界----%d¥n", __alignof__(e));
    printf("fooの境界----%d¥n", __alignof__(foo1));
    printf("foo.eの境界----%d¥n", __alignof__(foo1.e));
    return 0;
}
```

〔リスト 11〕**型あるいはアラインメントを問い合わせたソースから生成されたアセンブラ**(test91.s)

```
    .file       "test91.c"
    .version    "01.01"
gcc2 compiled.:
.section    .rodata
.LC0:
    .string "double¥244¥316¥266¥255¥263¥246----%d¥n"
.LC1:
    .string "float¥244¥316¥266¥255¥263¥246----%d¥n"
.LC2:
    .string "long long¥244¥316¥266¥255¥263¥246----%d¥n"
.LC3:
    .string "int¥244¥316¥266¥255¥263¥246----%d¥n"
.LC4:
    .string "char¥244¥316¥266¥255¥263¥246----%d¥n"
.LC5:
    .string "foo¥244¥316¥266¥255¥263¥246----%d¥n"
.LC6:
    .string "foo.e¥244¥316¥266¥255¥263¥246----%d¥n"
.text
    .align  4
.globl  main
    .type   main,@function
main:
    pushl   %ebp
    movl    %esp,%ebp
    subl    $8,%esp
    addl    $-8,%esp
    pushl   $4
    pushl   $.LC0
    call    printf
    addl    $-8,%esp
    pushl   $4
    pushl   $.LC1
    call    printf
    addl    $32,%esp
    addl    $-8,%esp
    pushl   $4
    pushl   $.LC2
    call    printf
    addl    $-8,%esp
    pushl   $4
    pushl   $.LC3
    call    printf
    addl    $32,%esp
    addl    $-8,%esp
    pushl   $1
    pushl   $.LC4
    call    printf
    addl    $-8,%esp
    pushl   $4
    pushl   $.LC5
    call    printf
    addl    $32,%esp
    addl    $-8,%esp
    pushl   $1
    pushl   $.LC6
    call    printf
    xorl    %eax,%eax
    movl    %ebp,%esp
    popl    %ebp
    ret
.Lfe1:
    .size   main,.Lfe1-main
    .ident  "GCC: (GNU) 2.95.3 20010315 (release)"
```

〔リスト12〕test91.cをSHプロセッサ用にコンパイルした際のアセンブラ(SH_test91.s)

```
    .file   "test.c"
    .data
gcc2_compiled.:
___gnu_compiled_c:
    .text
    .align 2
LC0:
    .ascii "double¥244¥316¥266¥255¥263¥246----%d¥12¥0"
    .align 2
LC1:
    .ascii "float¥244¥316¥266¥255¥263¥246----%d¥12¥0"
    .align 2
LC2:
    .ascii "long long¥244¥316¥266¥255¥263¥246----%d¥12¥0"
    .align 2
LC3:
    .ascii "int¥244¥316¥266¥255¥263¥246----%d¥12¥0"
    .align 2
LC4:
    .ascii "char¥244¥316¥266¥255¥263¥246----%d¥12¥0"
    .align 2
LC5:
    .ascii "foo¥244¥316¥266¥255¥263¥246----%d¥12¥0"
    .align 2
LC6:
    .ascii "foo.e¥244¥316¥266¥255¥263¥246----%d¥12¥0"
    .align 2
    .global  _main
_main:
    mov.l   r8,@-r15
    mov.l   L3,r1
    mov.l   r14,@-r15
    sts.l   pr,@-r15
    jsr     @r1
    mov     r15,r14
    mov.l   L4,r8
    mov.l   L5,r4
    jsr     @r8
    mov     #4,r5
    mov.l   L6,r4
    jsr     @r8
    mov     #4,r5
    mov.l   L7,r4
    jsr     @r8
    mov     #4,r5
    mov.l   L8,r4
    jsr     @r8
    mov     #4,r5
    mov.l   L9,r4
    jsr     @r8
    mov     #1,r5
    mov.l   L10,r4
    jsr     @r8
    mov     #4,r5
    mov.l   L11,r4
    jsr     @r8
    mov     #1,r5
    mov     #0,r0
    mov     r14,r15
    lds.l   @r15+,pr
    mov.l   @r15+,r14
    rts
    mov.l   @r15+,r8
L12:
    .align 2
L3:
    .long   ___main
L4:
    .long   _printf
L5:
    .long   LC0
L6:
    .long   LC1
L7:
    .long   LC2
L8:
    .long   LC3
L9:
    .long   LC4
L10:
    .long   LC5
L11:
    .long   LC6
```

複数の属性を指定するには，たとえば“__attribute__((aligned (16), packed))”のように，2重の丸括弧(())の中で属性をカンマで区切ります．

それぞれのキーワードの前後に__を付けて属性を指定することもできます．もし同じ名前をもつマクロが定義済みでも，ヘッダファイルの中でキーワードを使うことができるようになります．たとえば，alignedの代わりに__aligned__を使うことができます．

○ **aligned (alignment)**

この属性は，変数または構造体フィールドで最小のアラインメントの値をバイト単位で指定します．以下のソースはそれぞれアラインメントを16バイト，32バイトにしたものです．

オブジェクトファイルのシンボルをnmコマンドで出力してみます．すると，**リスト13～リスト16**のようにfoo1のアラインメントが変わっています．

〔リスト13〕変数のアラインメントを16バイトにしたソース(test92.c)

```
#include <stdio.h>
//変数属性の指定 aligned (alignment)
struct foo { int x[20] __attribute__ ((aligned (16))); }foo1;
int main(void)
{
    foo1.x[0]   =   1;
    foo1.x[1]   =   2;
    printf("foo1の境界----%d¥n", __alignof__ (foo1));
    return 0;
}
```

〔リスト14〕変数のアラインメントを32バイトにしたソース(test93.c)

```
#include <stdio.h>
//変数属性の指定 aligned (alignment)
struct foo { int x[20] __attribute__ ((aligned (32))); }foo1;
int main(void)
{
    foo1.x[0]   =   1;
    foo1.x[1]   =   2;
    printf("foo1の境界----%d¥n", __alignof__ (foo1));
    return 0;
}
```

〔リスト15〕変数のアラインメントを16バイトにしたソースをコンパイル後のオブジェクト配置(test92nm.txt)

```
08048334 t Letext
080494f8 ? _DYNAMIC
080494d8 ? _GLOBAL_OFFSET_TABLE_
080484a0 R _IO_stdin_used
080494cc ? __CTOR_END__
080494c8 ? __CTOR_LIST__
080494d4 ? __DTOR_END__
080494d0 ? __DTOR_LIST__
080494c4 ? __EH_FRAME_BEGIN__
080494c4 ? __FRAME_END__
08049598 A __bss_start
080494b8 D __data_start
         w __deregister_frame_info@@GLIBC_2.0
08048440 t __do_global_ctors_aux
08048360 t __do_global_dtors_aux
         w __gmon_start__
         U __libc_start_main@@GLIBC_2.0
         w __register_frame_info@@GLIBC_2.0
08049598 A _edata
08049610 A _end
08048480 ? _fini
0804849c R _fp_hw
08048298 ? _init
08048310 T _start
08048334 t call_gmon_start
080494c0 d completed.4
080494b8 W data_start
080483b4 t fini_dummy
080495c0 B foo1
080494c4 d force_to_data
080494c4 d force_to_data
080483c0 t frame_dummy
08048334 t gcc2_compiled.
08048360 t gcc2_compiled.
08048440 t gcc2_compiled.
08048480 t gcc2_compiled.
08048400 t gcc2_compiled.
080483e8 t init_dummy
08048468 t init_dummy
08048400 T main
080495a0 b object.11
080494bc d p.3
         U printf@@GLIBC_2.0
```

〔リスト 16〕変数のアラインメントを 32 バイトにしたソースコンパイル後のオブジェクト配置(test93nm.txt)

```
08048334 t Letext
080494f8 ? _DYNAMIC
080494d8 ? _GLOBAL_OFFSET_TABLE_
080484a0 R _IO_stdin_used
080494cc ? __CTOR_END__
080494c8 ? __CTOR_LIST__
080494d4 ? __DTOR_END__
080494d0 ? __DTOR_LIST__
080494c4 ? __EH_FRAME_BEGIN__
080494c4 ? __FRAME_END__
08049598 A __bss_start
080494b8 D __data_start
         w __deregister_frame_info@@GLIBC_2.0
08048440 t __do_global_ctors_aux
08048360 t __do_global_dtors_aux
         w __gmon_start__
         U __libc_start_main@@GLIBC_2.0
         w __register_frame_info@@GLIBC_2.0
08049598 A _edata
08049620 A _end
08048480 ? _fini
0804849c R _fp_hw
08048298 ? _init
08048310 T _start
08048334 t call_gmon_start
080494c0 d completed.4
080494b8 W data_start
080483b4 t fini_dummy
080495c0 B foo1
080494c4 d force_to_data
080494c4 d force_to_data
080483c0 t frame_dummy
08048334 t gcc2_compiled.
08048360 t gcc2_compiled.
08048440 t gcc2_compiled.
08048480 t gcc2_compiled.
08048400 t gcc2_compiled.
080483e8 t init_dummy
08048468 t init_dummy
08048400 T main
080495a0 b object.11
080494bc d p.3
         U printf@@GLIBC_2.0
```

〔リスト 17〕変数のアラインメントを自動的に設定したソース(test94.c)

```
#include <stdio.h>
//変数属性の指定 aligned (alignment)
struct foo { int x[20] __attribute__ ((aligned )); }foo1;
int main(void)
{
    foo1.x[0]   =   1;
    foo1.x[1]   =   2;
    printf("foo1の境界----%d¥n", __alignof__(foo1));
    return 0;
}
```

〔リスト 18〕mode 属性で変数の長さを変更しているソース(test95.c)

```
#include <stdio.h>
//変数属性の指定 mode (mode)
int main(void)
{
    char x __attribute__ ((mode(pointer) ));
    int y __attribute__ ((mode(byte) ));
    printf("xのサイズ----%d¥n", __alignof__(x));
    printf("yのサイズ----%d¥n", __alignof__(y));
    return 0;
}
```

〔リスト 19〕nocommon 属性を指定したソース(test96.c)

```
#include <stdio.h>
//変数属性の指定 nocommon
int x __attribute__ ((nocommon));
int main(void)
{
    printf("xの値----%d¥n",x);
    return 0;
}
```

〔リスト 20〕nocommon 属性を指定したソースから生成されたアセンブラ(test96.s)

```
    .file       "test96.c"
    .version    "01.01"
gcc2_compiled.:
.section    .rodata
.LC0:
    .string "x¥244¥316¥303¥315----%d¥n"
.text
    .align  4
.globl  main
    .type   main,@function
main:
    pushl   %ebp
    movl    %esp,%ebp
    subl    $8,%esp
    addl    $-8,%esp
    movl    x,%eax
    pushl   %eax
    pushl   $.LC0
    call    printf
    addl    $16,%esp
    xorl    %eax,%eax
    jmp .L2
    .p2align    4,,7
.L2:
    movl    %ebp,%esp
    popl    %ebp
    ret
.Lfe1:
    .size   main,.Lfe1-main
.globl  x
.bss
    .align  4
    .type   x,@object
    .size   x,4
x:
    .zero   4
    .ident  "GCC: (GNU) 2.95.3 20010315 (release)"
```

なお，aligned 属性の指定においてアラインメントの係数を省略した場合，コンパイルのターゲットマシン上においてもっとも効率のよい値をコンパイラが自動的に設定します．インテル x86 アーキテクチャの場合は，**リスト 17** のように int の場合は 4 が設定されます．

以下は実行結果です．

```
$ gcc  -o test94 test94.c
$ ./test94
foo1の境界----4
$
```

○ **mode (mode)**

この属性は，宣言のデータ型を指定します．指定される型は，モード mode に対応する型です．byte を指定すれば 1 バイトになります．word を指定すればその環境の 1 ワード，pointer を指定すればその環境で使用できるポインタの長さが確保されます(**リスト 18**)．

実行結果は次のようになります．

```
$ gcc  -o test95 test95.c
$ ./test95
xのサイズ----4
yのサイズ----1
$
```

リスト 18 では char 型にポインタの長さを，int 型に 1 バイトを指定しています．プロセッサの環境が変わってもポインタの長さを正確に確保したり，1 ワードの大きさを正確に確保する場合に有効な方法ですが，GCC に慣れていない人には理解でき

〔リスト21〕**packed属性を指定したソース**（test97.c）

```
#include <stdio.h>
//変数属性の指定 packed
struct foo
{
    char a;
    int x[2] __attribute__ ((packed));
}foo1;
struct foo
{
    char a;
    int x[2];
}foo1_;
int main(void)
{
    foo1.a  =   0x00;
    foo1.x[0]   =   0x00;
    printf("foo1のサイズ----%d¥n",sizeof(foo1));
    printf("foo1.xのサイズ----%d¥n",sizeof(foo1.x));
    printf("foo1_のサイズ----%d¥n",sizeof(foo1_));
    printf("foo1_.xのサイズ----%d¥n",sizeof(foo1_.x));
    return 0;
}
```

〔リスト22〕**packed属性を指定したソースから生成されたアセンブラ**（test97.s）

```
    .file       "test97.c"
    .version    "01.01"
gcc2 compiled.:
.section    .rodata
.LC0:
    .string "foo1¥244¥316¥245¥265¥245¥244¥245¥272----%d¥n"
.LC1:
    .string "foo1.x¥244¥316¥245¥265¥245¥244¥245¥272----%d¥n"
.LC2:
    .string "foo1_¥244¥316¥245¥265¥245¥244¥245¥272----%d¥n"
.LC3:
    .string "foo1_.x¥244¥316¥245¥265¥245¥244¥245¥272----%d¥n"
.text
    .align  4
.globl  main
    .type   main,@function
main:
    pushl   %ebp
    movl    %esp,%ebp
    subl    $8,%esp
    movb    $0,foo1
    movl    $0,foo1+1
    addl    $-8,%esp
    pushl   $9
    pushl   $.LC0
    call    printf
    addl    $16,%esp
    addl    $-8,%esp
    pushl   $8
    pushl   $.LC1
    call    printf
    addl    $16,%esp
    addl    $-8,%esp
    pushl   $12
    pushl   $.LC2
    call    printf
    addl    $16,%esp
    addl    $-8,%esp
    pushl   $8
    pushl   $.LC3
    call    printf
    addl    $16,%esp
    xorl    %eax,%eax
    jmp .L2
    .p2align    4,,7
.L2:
    movl    %ebp,%esp
    popl    %ebp
    ret
.Lfe1:
    .size   main,.Lfe1-main
    .comm   foo1,9,1
    .comm   foo1_,12,4
    .ident  "GCC: (GNU) 2.95.3 20010315 (release)"
```

ないでしょう．

○ **nocommon**

この属性を指定するとグローバル変数の場合，.bssセクションに配置します（**リスト19**，**リスト20**）．なお，その変数は初期化されます．

○ **packed**

packed属性は，aligned属性が指定されていない限り，変数または構造体フィールドが，アラインメントを可能な限り最小の値にすることを指定します．この最小値は，変数についてはlバイトであり，フィールドについては1ビットです（**リスト21**，**リスト22**）．

実行結果は次のようになります．

```
$ ./test97
foo1のサイズ----9
foo1.xのサイズ----8
foo1_のサイズ----12
foo1_.xのサイズ----8
$
```

○ **section（"section-name"）**

コンパイルの際に，通常は生成するオブジェクトをdataやbssといったセクションに置きます．しかし，場合によっては，たとえば特殊なハードウェアにマップするために，追加のセクションが必要になったり，特定の変数を特殊なセクションに置くことが必要になります．

section属性は，ある変数が，ある特定のセクション内に存在するよう指定します（**リスト23**，**リスト24**）．

このように，それぞれfoo1_section，foo2_section，data_sectionに置かれています．

オブジェクト上では**リスト25**のように配置されます．

○ **transparent_union**

この属性は共用体型の関数パラメータに対して指定されます．そのパラメータに対応する引き数自体は，その共用体の任意のメンバの型をもつことができます．しかし，その引き数がその関数に渡される際には，共用体の1番目のメンバの型をもつものとして扱われます．詳細については，型属性の指定で説明します．

○ **unused**

この属性は変数に対して指定されます，その変数はおそらく使われないはずであるということを意味します．GNU Cは，その変数については警告メッセージを出力しません．

○ **weak**

weak属性については，第7回（本誌2003年3月号）の連載にある関数属性の宣言で説明しています．

○ **model（model-name）**

model属性はオブジェクトがスモール/ミディアム/ラージであると宣言しなくてはならない環境において，それを宣言するものです．

● 型属性の指定

キーワード__attribute__により，struct型とunion型を定義する際，その型に特殊な属性を指定することができます．このキーワードの後に，2重の丸括弧(())に囲まれた属性指定が続きます．現在，3個の属性aligned，packed，transparent_unionが型に対して定義されています．

属性はいずれも，キーワードの前後に__を付けて指定することもできます．これにより，同じ名前をもつマクロが定義済みであるかどうかを心配することなく，ヘッダファイルの中でこの属性を使うことができるようになります．たとえば，alignedの代わりに__aligned__を使うことができます．

aligned属性とtransparent_union属性は，typedef宣言の中において，あるいは完結した列挙型，構造体型，共用体型の定義の終端の波括弧 }の直後において指定することができます．また，packed属性は，定義の終端の波括弧 }の後においてのみ指定することができます．

また，終端の波括弧 }の後ではなく，列挙タグ，構造体タグ，共用体タグと型の名前の間において，属性を指定することもできます．

○ **aligned（alignment）**

この属性は，指定された型の変数の最小のアラインメントの値を(バイト単位で)指定します(**リスト26**)．

この場合，ダブルワード単位に転送することができるので，実行時の効率が向上します．

〔リスト23〕section属性を指定したソース（test98.c）

```
#include <stdio.h>
//変数属性の指定 section
struct foo
{
    char a;
    int x[2] ;
}foo __attribute__ ((section ("foo1 section") ));
struct foo2
{
    char a;
    int x[2];
}foo2 __attribute__ ((section ("foo2 section") ));
int data __attribute__ ((section ("data section") ));
int main(void)
{
    data    =   1;
    return 0;
}
```

〔リスト24〕section属性を指定したソースから生成されたアセンブラ（test98.s）

```
    .file       "test98.c"
    .version    "01.01"
gcc2 compiled.:
.text
    .align  4
.globl  main
    .type   main,@function
main:
    pushl   %ebp
    movl    %esp,%ebp
    movl    $1,data
    xorl    %eax,%eax
    jmp .L2
    .p2align    4,,7
.L2:
    movl    %ebp,%esp
    popl    %ebp
    ret
.Lfe1:
    .size   main,.Lfe1-main
.globl  foo
.section    foo1 section,"aw",@progbits
    .align  4
    .type   foo ,@object
    .size   foo ,12
foo :
    .zero   12
.globl  foo2
.section    foo2 section,"aw",@progbits
    .align  4
    .type   foo2 ,@object
    .size   foo2 ,12
foo2 :
    .zero   12
.globl  data
.section    data section,"aw",@progbits
    .align  4
    .type   data,@object
    .size   data,4
data:
    .zero   4
    .ident  "GCC: (GNU) 2.95.3 20010315 (release)"
```

〔リスト25〕section属性を指定したソースをリンクしたオブジェクト配置リスト（test98nm.txt）

```
08048304 t Letext
080494ac ? _DYNAMIC
08049490 ? _GLOBAL_OFFSET_TABLE_
08048450 R _IO_stdin_used
08049484 ? __CTOR_END__
08049480 ? __CTOR_LIST__
0804948c ? __DTOR_END__
08049488 ? __DTOR_LIST__
0804947c ? __EH_FRAME_BEGIN__
0804947c ? __FRAME_END__
0804954c A __bss_start
08049454 D __data_start
         w __deregister_frame_info@@GLIBC_2.0
080483f0 t __do_global_ctors_aux
08048330 t __do_global_dtors_aux
         w __gmon_start__
         U __libc_start_main@@GLIBC_2.0
         w __register_frame_info@@GLIBC_2.0
08049478 A __start_data_section
08049460 A __start_foo1_section
0804946c A __start_foo2_section
0804947c A __stop_data_section
0804946c A __stop_foo1_section
08049478 A __stop_foo2_section
0804954c A _edata
08049564 A _end
08048430 ? _fini
0804844c R _fp_hw
08048274 ? _init
080482e0 T _start
08048304 t call_gmon_start
0804945c d completed.4
08049478 ? data
08049454 W data_start
08048384 t fini_dummy
0804946c ? foo2
08049460 ? foo
08049460 d force_to_data
08049460 d force_to_data
08048390 t frame_dummy
08048304 t gcc2_compiled.
08048330 t gcc2_compiled.
080483f0 t gcc2_compiled.
08048430 t gcc2_compiled.
080483d0 t gcc2_compiled.
080483b8 t init_dummy
08048418 t init_dummy
080483d0 T main
0804954c b object.11
08049458 d p.3
```

必要に応じてアラインメントの数値を指定できますが，リンカによっては制限がある場合があります．

リスト27 のように eax レジスタと ax レジスタで，それぞれ32ビット転送と16ビット転送を行っています．

○ packed

前項「変数の属性の指定」で packed を説明しましたが，型の指定でこれを行うと変数ではなく，その型全体が packed 属性になります．つまり struct 型と union 型に対してこの属性を指定するのは，構造体または共用体の個々のメンバに packed 属性を指定するのと同等です．

○ transparent_union

名前のとおり，透過性共用体とでもいいましょうか．union の型定義にこれが指定された場合，その共用体の型をもつ関数パラメータがあると，その関数の呼び出しが特殊な方法で扱われるようになります．

これには，大きく分けて二つの機能があります．

一つ目の機能：transparent_union 属性が指定された共用体型に対応する引き数は，その共用体に定義された任意のメンバの型をもつことができます．キャストは必要ありません．また，その共用体のメンバにポインタ型のものがあれば，対応する引き数にはヌルポインタ定数または void ポインタ式を使うことができます．

なお，その共用体のメンバに void ポインタ型のものがあれば，対応する引き数には任意のポインタ式を使うことができます．

二つ目の機能：関数に対してその引き数が渡される際には，transparent_union 属性が指定された共用体自体の呼び出し規約ではなく，その共用体の1番目のメンバの呼び出し規約が使われます．

この機能は次のような場合に有用です．たとえば，互換性のために複数のインターフェースをもつライブラリ関数に使用できます．くわしく説明すると，wait 関数は POSIX との互換のために int ＊型の値を受け付けなければなりません．その一方で，4.1BSD インターフェースとの互換のために union wait ＊型の値を受け付けなければならないのです．

もし wait のパラメータが void ＊型であったとすると，wait はどちらの引き数も受け付けるでしょうが，その他の任意のポインタ型も受け付けることになってしまい，引き数の型チェックがあまり役に立たなくなるでしょう．こうする代わりに<sys/wait.h>では，そのインターフェースを次のように定義することができます．

```
typedef union
  {
    int *__ip;
    union wait *__up;
  } wait_status_ptr_t __attribute__
                ((__transparent_union__));
```

〔リスト26〕aligned を指定したソース(test100.c)

```
#include <stdio.h>
//型属性の指定 aligned (alignment)
struct data1 { short f[3]; } __attribute__ ((aligned (8)));
struct data2 { short f[3]; };
struct data1 dataa__;
struct data1 datab__;
struct data2 datac__;
struct data2 datad__;
int main(void)
{
    asm("nop");
    datab__  =  dataa__;
    asm("nop");
    datad__  =  datac__;
    asm("nop");
    return 0;
}
```

〔リスト27〕aligned を指定したソースから生成したアセンブラ(test100.s)

```
    .file       "test100.c"
    .version    "01.01"
gcc2 compiled.:
.text
    .align  4
.globl  main
    .type   main,@function
main:
    pushl   %ebp
    movl    %esp,%ebp
#APP
    nop
#NO APP
    movl    dataa__,%eax
    movl    %eax,datab__
    movl    dataa__+4,%eax
    movl    %eax,datab__+4
#APP
    nop
#NO APP
    movl    datac__,%eax
    movl    %eax,datad__
    movw    datac__+4,%ax
    movw    %ax,datad__+4
#APP
    nop
#NO APP
    xorl    %eax,%eax
    jmp .L2
    .p2align    4,,7
.L2:
    movl    %ebp,%esp
    popl    %ebp
    ret
.Lfe1:
    .size   main,.Lfe1-main
    .comm   dataa__,8,8
    .comm   datab__,8,8
    .comm   datac__,6,2
    .comm   datad__,6,2
    .ident  "GCC: (GNU) 2.95.3 20010315 (release)"
```

```
pid_t wait (wait_status_ptr_t);
```

このインターフェースでは，int ＊型と union wait ＊型の引き数どちらでも渡すことができます．

```
int w1 () { int w; return wait (&w); }
int w2 () { union wait w; return wait (&w); }
```

このインターフェースでは，wait の実装例は次のようになります．

```
pid_t wait (wait_status_ptr_t p)
{
```

〔リスト 28〕asm 命令の operand constraint string 概略を説明する C ソース(test105.c)

```
#include <stdio.h>
//C の式をオペランドとして持つアセンブラ命令
int main(void)
{
    int x   =   100;
    int y   =   500;
    int result;
    asm("nop");
//#割り当て場所はメモリでもレジスタでも構わない設定
    asm("movl  %0,  %%eax" :: "g"(x): "ax");
    asm("movl  %0,  %%ebx" :: "g"(y): "bx");
    asm("addl  %%ebx, %%eax" ::: "ax","bx");
    asm("movl  %%eax, %0"  : "=g"(result) :: "ax");
    asm("nop");
//割り当て場所はレジスタである
    asm("movl  %0,  %%eax" :: "r"(x): "ax");
    asm("movl  %0,  %%ebx" :: "r"(y): "bx");
    asm("addl  %%ebx, %%eax" ::: "ax","bx");
    asm("movl  %%eax, %0"  : "=r"(result) :: "ax");
    asm("nop");
//割り当て場所はメモリである
    asm("movl  %0,  %%eax" :: "m"(x): "ax");
    asm("movl  %0,  %%ebx" :: "m"(y): "bx");
    asm("addl  %%ebx, %%eax" ::: "ax","bx");
    asm("movl  %%eax, %0"  : "=m"(result) :: "ax");
    asm("nop");
//割り当て場所は EAX, EBX, ECX, EDX レジスタのいずれか
    asm("movl  %0,  %%eax" :: "q"(x): "ax");
    asm("movl  %0,  %%ebx" :: "q"(y): "bx");
    asm("addl  %%ebx, %%eax" ::: "ax","bx");
    asm("movl  %%eax, %0"  : "=q"(result) :: "ax");
    asm("nop");
//
    asm("movl  %0,  %%eax" :: "D"(x): "ax");          //EDI レジスタに割り当て指定
    asm("movl  %0,  %%ebx" :: "S"(y): "bx");          //ESI レジスタに割り当て指定
    asm("addl  %%ebx, %%eax" ::: "ax","bx");
    asm("movl  %%eax, %0"  : "=c"(result) :: "ax"); //ECX レジスタに割り当て指定
    asm("nop");
    printf("%d¥n",result);
    return 0;
  }
```

```
    return waitpid (-1, p.__ip, 0);
  }
```

○ **unused**

前項と同じように，この属性が型に対して指定されると，その型をもつ変数はおそらく使われないはずであるということを意味します．GNU C は，その変数については警告メッセージを出力しません．

● Cの式をオペランドとしてもつアセンブラ命令

asm を使ったアセンブラ命令の中でオペランドを C の式を使って指定することができます．このことは，使いたいデータがどのレジスタまたはメモリ位置に保持されるのかを推測する必要がないということを意味しています．

マシン記述(machine description)の中で使われるものとよく似たアセンブラ命令テンプレートに加えて，個々のオペランドの operand constraint string を指定しなければなりません．

拡張されたアセンブラ命令の構文は次のとおりです．

asm (アセンブリコード : 出力オペランド : 入力オペランド : 保持されないレジスタ);

○ **operand constraint string について**

出力オペランドや入力オペランド中で，それに続く括弧でくくった式に，どのようなレジスタやメモリを割り当てるかを決めます．この概略は以下に記します．

○ **アセンブリコードについて**

内容が破損してしまうレジスタを3番目の : の後に書いておくと，レジスタの退避，復帰のためのコードを自動的に生成してくれます．プログラマはレジスタの退避，復帰を省いて書くことができます．

○ **operand constraint string についての説明**

x86 系でよく使われる制約は次のとおりです．

“r”: そのオペランドがレジスタでなくてはならないことを表します

“f”: 浮動小数点レジスタでなくてはならないことを表します

“m”: メモリオペランドでなくてはならないことを表します

“g”: メモリでもレジスタでも構わないときにこれを指定します

“q”: EAX, EBX, ECX, EDX レジスタに割り当てることを意味します

“a”,“b”,“c”,“d”,“D”,“S”,“B”:
それぞれ，EAX, EBX, ECX, EDX, EDI, ESI, EBP の各レジスタに割り当てることを意味します

“0”,“1”, ……:
これらの数字は入力オペランドのみに指定できます．出力オペランド %0 や %1 と同一であることをコンパイラに教えることができます

例として単純な足し算を行うプログラムソースを**リスト 28**, **リスト 29** に掲載します．

これらのリストのとおり，制約を指定するとそのとおりに変数の割り当てを行います．

なお，これは Intel 386 系に関する情報です．ほかのプロセッサについても，いろいろな operand constraint string があります．ARM ファミリ，AMD 29000 ファミリ，IBM RS6000, Intel 960, MIPS, Motorola 680x0, SPARC ……以上のプロセッサ特有の設定があるので，詳細は GCC マニュアルを参照してください．

● アセンブラコードの中で使われる名前の制御

C の関数または変数に対してアセンブラコードの中で使われる名前を，以下のようにその宣言子の後に asm(または __asm__) キーワードを書くことによって，指定することができます(**リスト 30**, **リスト 31**)．

このように，アセンブラ中での名前が変更されています．

注意点としては，変更後の名前がアセンブラ中で衝突しないようにすることです．そうなってしまった場合，当然のことながらコンパイルエラーにはなりません．

● 指定されたレジスタの中の変数

GNU C では，指定されたハードウェアレジスタの中に少数の

〔リスト29〕asm命令のoperand constraint string概略を説明するCソースから生成したアセンブラ(test105.s)

```
    .file       "test105.c"
    .version    "01.01"
gcc2 compiled.:
.section    .rodata
.LC0:
    .string "%d¥n"
.text
    .align  4
.globl  main
    .type   main,@function
main:
    pushl   %ebp
    movl    %esp,%ebp
    subl    $28,%esp
    pushl   %edi
    pushl   %esi
    pushl   %ebx
    movl    $100,-4(%ebp)
    movl    $500,-8(%ebp)
#APP
#割り当て場所はメモリでもレジスタでも構わない設定
    nop
    movl    -4(%ebp),%eax
    movl    -8(%ebp),%ebx
    addl    %ebx,%eax
    movl    %eax,-12(%ebp)
    nop
#NO APP
    movl    -4(%ebp),%edx
#APP
#割り当て場所はレジスタである
    movl    %edx,%eax
#NO APP
    movl    -8(%ebp),%edx
#APP
    movl    %edx,%ebx
    addl    %ebx,%eax
    movl    %eax,%edx
#NO APP
    movl    %edx,%eax
    movl    %eax,-12(%ebp)
#APP
    nop
#割り当て場所はメモリである
    movl    -4(%ebp),%eax
    movl    -8(%ebp),%ebx
    addl    %ebx,%eax
    movl    %eax,-12(%ebp)
    nop
#NO APP
    movl    -4(%ebp),%edx
#APP
#割り当て場所はEAX, EBX, ECX, EDXレジスタのいずれか
    movl    %edx,%eax
#NO APP
    movl    -8(%ebp),%edx
#APP
    movl    %edx,%ebx
    addl    %ebx,%eax
    movl    %eax,%edx
#NO APP
    movl    %edx,%eax
    movl    %eax,-12(%ebp)
#APP
    nop
#NO APP
    movl    -4(%ebp),%edi
#APP
    movl    %edi,%eax   #EDIレジスタに割り当て指定
#NO APP
    movl    -8(%ebp),%esi
#APP
    movl    %esi,%ebx   #ESIレジスタに割り当て指定
    addl    %ebx,%eax
    movl    %eax,%ecx   #ECXレジスタに割り当て指定
#NO APP
    movl    %ecx,%eax
    movl    %eax,-12(%ebp)
#APP
    nop
#NO APP
    addl    $-8,%esp
    movl    -12(%ebp),%eax
    pushl   %eax
    pushl   $.LC0
    call    printf
    addl    $16,%esp
    xorl    %eax,%eax
    jmp .L2
    .p2align    4,,7
.L2:
    leal    -40(%ebp),%esp
    popl    %ebx
    popl    %esi
    popl    %edi
    movl    %ebp,%esp
    popl    %ebp
    ret
.Lfe1:
    .size   main,.Lfe1-main
    .ident  "GCC: (GNU) 2.95.3 20010315 (release)"
```

〔リスト30〕アセンブラ中で名前を変更した例のCソース(test101.c)

```
#include <stdio.h>
int hensu[10] asm ("hensu asm");
int tbl[10];
extern int  kansu () asm ("asm kansu");

int main(void)
{
    hensu[0]    =   1;
    tbl[0]      =   kansu();
    return 0;
}
int kansu (void)
{
    return  0;
}
```

〔リスト31〕アセンブラ中で名前を変更した例のCソースから生成したアセンブラ(test101.s)

```
    .file       "test101.c"
    .version    "01.01"
gcc2 compiled.:
.text
    .align  4
.globl  main
    .type   main,@function
main:
    pushl   %ebp
    movl    %esp,%ebp
    subl    $8,%esp
    movl    $1,hensu asm
    call    asm kansu
    movl    %eax,%eax
    movl    %eax,tbl
    xorl    %eax,%eax
    jmp .L2
    .p2align    4,,7
.L2:
    movl    %ebp,%esp
    popl    %ebp
    ret
.Lfe1:
    .size   main,.Lfe1-main
    .align  4
.globl  asm kansu
    .type   asm kansu,@function
asm kansu:
    pushl   %ebp
    movl    %esp,%ebp
    xorl    %eax,%eax
    jmp .L3
    .p2align    4,,7
.L3:
    movl    %ebp,%esp
    popl    %ebp
    ret
.Lfe2:
    .size   asm kansu,.Lfe2-asm kansu
    .comm   hensu asm,40,32
    .comm   tbl,40,32
    .ident  "GCC: (GNU) 2.95.3 20010315 (release)"
```

広域変数を置くことができます．また，通常のレジスタ変数が割り当てられるべきレジスタを指定することもできます．しかし，めったに使わない機能だと思います．

1）広域レジスタ変数の定義

GNU Cでは**リスト32**，**リスト33**のようにして，広域レジスタ変数を定義することができます．

問題は，プロセッサによってレジスタの名称が異なるため，環境に依存してしまうことです．それを理解して使用するのであれば，効率の良いコードを書くことができるでしょう．

2）局所変数に対するレジスタの指定

同じように局所変数もレジスタに割り当てることができます．宣言の場所が違うだけです．同じように環境に依存します（**リスト34**，**リスト35**）．

● 代替キーワード

オプション`-traditional`を使うと，特定のキーワードが利用できなくなります．オプション`-ansi`を使うと，別の特定のキーワードが利用できなくなります．

ANSI Cのプログラムや伝統的なCのプログラムも含むすべてのプログラムにおいて，利用可能でなければならない汎用的なヘッダファイルの中で，GNU Cの拡張機能やANSI Cの機能を使いたい場合に，これが問題になります．

キーワード`asm`，`typeof`，`inline`は，`-ansi`を指定してコンパイルされるプログラムの中では問題があり，キーワード`const`，`volatile`，`signed`，`typeof`，`inline`は，`-traditional`を指定してコンパイルされるプログラムの中では問題があります．

この問題を解決する方法は，問題のある個々のキーワードの前後に`__`を付けることです．たとえば，`asm`の代わりに`__asm__`を，`const`の代わりに`__const__`を，`inline`の代わりに`__inline__`を使ってください．

これはGNU Cの拡張機能なので，ほかのコンパイラではエラーとなってしまいます．ほかのコンパイラでコンパイルを行いたいのであれば，代替キーワードをマクロとして定義して，慣習的なキーワードと置き換えることができます．それは，次のようになります．

```
#ifndef __GNUC__
#define __asm__ asm
#endif
```

`-pedantic`を指定すると，ほとんどのGNU C拡張機能に対して警告メッセージが出力されます．式の前に`__extension__`と書くことによって，ある式の中における警告メッセージの出力を防ぐことができます．`__extension__`にはこれ以外の作用はありません．

● 不完全なenum型

`enum`タグを，それがもつことのできる値を指定せずに定義することができます．これはプログラムの意味がわかりにくくなってしまうと思います．`enum`の処理と`struct`や`union`の処理の一貫性がより高くなるという利点はありますが，あまり使わないほうがよいかもしれません．

● 関数名の文字列

カレントな関数の名前を値としてもつ二つの文字列変数があらかじめ定義されています．変数`__FUNCTION__`は，ソースコードの中に記述されたとおりの関数名です．一方，変数`__PRETTY_FUNCTION__`は，言語固有のスタイルに変更され

〔リスト32〕広域レジスタ変数の定義を行った例のCソース（test102.c）

```
#include <stdio.h>
//広域レジスタ変数の定義
register int *foo1 asm ("ebx");
register int *foo2 asm ("edx");
int hensu[10] asm ("hensu_asm");
int tbl[10];
extern int  kansu (void) asm ("asm_kansu");

int main(void)
{
    *foo1   =   100;
    *foo2   =   200;
    hensu[0]    =   1;
    tbl[0]      =   kansu();
    return 0;
}
int kansu (void)
{
    return  *foo1 * *foo2;
}
```

〔リスト33〕広域レジスタ変数の定義を行った例のCソースから生成したアセンブラ（test102.s）

```
    .file       "test102.c"
    .version    "01.01"
gcc2 compiled.:
.text
    .align  4
.globl  main
    .type   main,@function
main:
    pushl   %ebp
    movl    %esp,%ebp
    subl    $8,%esp
    movl    $100,(%ebx)
    movl    $200,(%edx)
    movl    $1,hensu_asm
    call    asm_kansu
    movl    %eax,%eax
    movl    %eax,tbl
    xorl    %eax,%eax
    jmp     .L2
    .p2align    4,,7
.L2:
    movl    %ebp,%esp
    popl    %ebp
    ret
.Lfe1:
    .size   main,.Lfe1-main
    .align  4
.globl  asm_kansu
    .type   asm_kansu,@function
asm_kansu:
    pushl   %ebp
    movl    %esp,%ebp
    movl    (%ebx),%ecx
    imull   (%edx),%ecx
    movl    %ecx,%eax
    jmp .L3
    .p2align    4,,7
.L3:
    movl    %ebp,%esp
    popl    %ebp
    ret
.Lfe2:
    .size   asm_kansu,.Lfe2-asm_kansu
    .comm   hensu_asm,40,32
    .comm   tbl,40,32
    .ident  "GCC: (GNU) 2.95.3 20010315 (release)"
```

〔リスト34〕局所レジスタ変数の定義を行った例のCソース(test103.c)

```
#include <stdio.h>
//局所レジスタ変数の定義
int hensu[10] asm ("hensu asm");
int tbl[10];
int main(void)
{
    register int *foo1 asm ("eax");
    register int *foo2 asm ("ebx");
    *foo1   =   100;
    *foo2   =   200;
    hensu[0]    =   1;
    tbl[0]      =   *foo1   *   *foo2;
    return 0;
}
```

た関数名です.

C言語では，この二つは同じになります(**リスト36**).

```
$ gcc  -o test104 test104.c
$ ./test104
__FUNCTION__ = main
__PRETTY_FUNCTION__ = main
$
```

● 関数の復帰アドレスとフレームアドレスの獲得

以下の関数を使うことで，復帰アドレス，フレームアドレスを取得できます.

○ **__builtin_return_address (level)**

この関数は，実行中の関数の復帰アドレス，または，実行中の関数を呼び出すまでに，途中で呼び出されてきた関数の中の一つの復帰アドレスを返します.

引き数levelは，呼び出しスタック中においてさかのぼるべきフレームの数です．値0を指定すると，実行中の関数の復帰アドレスが返ってきます．値1を指定すると，実行中の関数を呼び出した関数の復帰アドレスが返ってきます．以下，同様です.

引き数levelは整数の定数でなければなりません．マシンの中には，実行中の関数以外の関数の復帰アドレスを決定することが不可能なものがあります．そのような場合，あるいは，スタックのトップに達してしまった場合には，この関数は0を返します．この関数をデバッグの目的で使う際には，引き数には0以外の値だけを指定するべきです.

○ **__builtin_frame_address (level)**

この関数は，__builtin_return_addressに似ていますが，関数の復帰アドレスではなく関数フレームのアドレスを返します．値0を指定して__builtin_frame_addressを呼び出すと，実行中の関数のフレームアドレスが返ってきます．値1を指定すると，実行中の関数を呼び出した関数のフレームアドレスが返ってきます．以下，同様です.

フレームとは，局所変数や待避されたレジスタを保持している，スタック上の領域のことです．通常，フレームアドレスとは，関数によって最初にスタックにプッシュされたワードのアドレスのことです．しかし，正確な定義は，プロセッサと呼び出し規約に依存します．プロセッサが専用のフレームポインタレジスタをもつ場合，関数がフレームをもっていると，__builtin_frame_addressはフレームポインタレジスタの値を返します．__builtin_return_addressにあてはまる注意事項は，この関数にもあてはまります.

リンカの出力するオブジェクト配置リストなどを使用し，デバッグのためにこの関数を使用することで，効率の良いデバッグが可能になります.

〔リスト35〕局所レジスタ変数の定義を行った例のCソースから生成したアセンブラ(test103.s)

```
    .file       "test103.c"
    .version    "01.01"
gcc2 compiled.:
.text
    .align  4
.globl  main
    .type   main,@function
main:
    pushl   %ebp
    movl    %esp,%ebp
    pushl   %ebx
    movl    $100,(%eax)
    movl    $200,(%ebx)
    movl    $1,hensu asm
    movl    (%eax),%eax
    imull   (%ebx),%eax
    movl    %eax,tbl
    xorl    %eax,%eax
    jmp     .L2
.L2:
    movl    -4(%ebp),%ebx
    movl    %ebp,%esp
    popl    %ebp
    ret
.Lfe1:
    .size   main,.Lfe1-main
    .comm   hensu asm,40,32
    .comm   tbl,40,32
    .ident  "GCC: (GNU) 2.95.3 20010315 (release)"
```

〔リスト36〕関数名の文字列変数の例のCソース(test104.c)

```
#include <stdio.h>
//関数名の文字列
int main(void)
{
    printf ("__FUNCTION__ = %s¥n", __FUNCTION__);
    printf ("__PRETTY_FUNCTION__ = %s¥n", __PRETTY_FUNCTION__);
    return 0;
}
```

おわりに

さて，拡張機能についてはこれで終わりです.

次回は「ISO/IEC 9899：1999 ―― Programming Language C」(略称：C99)規格について，説明と検証を行う予定です.

参考文献

1) GCCマニュアル，Free Software Foundation

きし・てつお

● 活版印刷の発明

時は15世紀中ごろ，ルネッサンスの真っただ中にあるドイツで，グーテンベルグが後に世界三大発明の一つといわれる活版印刷を発明した．その後，弟子のフストとシェッファーは，ページ番号振り，奥付(執筆者などを書くページ)，脚注，色刷りなど，現在でも使われている本の体裁を短期間で確立してしまった．これは驚くべきことだ．当時，いかに出版に対する需要が高かったのかを知ることができる．

この技術の高さを証明するものは，意外にも日本にあった．16世紀末，ローマ法王に謁見した天正少年使節が，グーテンベルグの流れをくむ活版印刷機を日本に持ち帰り，キリシタン版といわれる出版をしていたのだ．「伊曾附(イソップ)物語」など30種類ほどが現存するが，現在の出版技術からみても，その完成度に対する評価は高い．惜しむらくは，キリシタン禁制で出版は禁止，以後二度と再開されなかったことだ．

一方，東洋には古くから木版(後に銅版)印刷という印刷技術があった．これは木版に多くの文字を彫って版画のように印刷する凹版印刷の一種だ．後漢の時代，碑文などの拓本を取ることから発達したといわれている．法隆寺には，8世紀後半といわれる世界最古の木版印刷物「百万塔陀羅尼」がある．

その後，中国では13世紀に王禎(おうてい)が，木の円盤の周りに複数の文字を彫った活字を使って「農書」を印刷している．この活版印刷技術は，豊臣秀吉の朝鮮出兵の副産物として日本に持ち帰られ，桃山文化とともに活版印刷が一気に普及する．後に徳川家康は，銅製の駿河版活字(重要文化財)を作成し，「群書治要」などいくつかの出版を行った．しかし，家康の死後，活字出版は急速に消えた．活版印刷は日本には根付かなかったのだ．その代わりに，仏典などの印刷に使われていた，古くからある木版印刷が復活し，明治まで続くことになる．

ここまで読んだ方はおわかりかもしれない．本格的な印刷は，じつは東洋で始まり進歩したのだ．そして14世紀中ごろに西洋に伝わったといわれる．グーテンベルグの活版印刷の発明は，中国の王禎の200年も後だった．しかし，西洋では印刷技術が著しい発展をみる．一方，残念なことに東洋ではさほど進展しなかった．それには二つの理由がある．

● 東洋と西洋の読書文化の違い

その理由の一つは，もともと東洋の文字は，活字を使う活版印刷より木版印刷が向いているということだ．たとえば，西洋の文字種は数十程度だ．しかも，複雑な文字はない．これに対し，東洋の文字種は数千を超え，文字自体もたいへん複雑だ．これでは，活字を作り，維持するのに膨大な労力がかかる．

そしてもう一つ，最大の理由がある．東洋には活版印刷の必要性が少なかったということだ．「読書百遍」という言葉をご存知だろう．これは魏の明帝の時代に高僧がいった言葉として記録に残っている．「必ず読むこと読書百遍なるべしと．そのこころは読書百遍にして義自ら見るればなり」(魏志)．ここで読書の「書」とは，経文を指している．

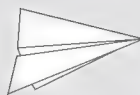
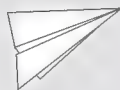
この言葉にあるように，東洋では，古くから経典や秘伝書を部屋に持ち込み，熟読して奥義を理解するという習慣を美徳とした．しかも当時の東洋はどこも海外との交易が少なく，時代の変化も遅かった．多品種大量印刷の必然性が失せていたのだ．

一方，西洋ではルネッサンス以降，大航海時代に突入，第2次産業革命が起きていた．宣教師が世界各国を飛び回り，聖書の解説書が多く作られ，次々と開発される最新技術を伝える本も大量に出版された．この違いが，東西の出版技術の発展に大きな差を生むことになった．

● 欧米に制覇された日本の出版技術

明治時代に入ると，西洋はアメリカを加えて欧米と呼ばれるようになった．活版が日本で復活したのは，ちょうどこの頃だ．本木昌造が，アメリカ人の指導をうけ，「横浜毎日新聞」(現毎日新聞)を刊行したのだ．以降，日本の出版技術は完全に欧米の指導のもと，欧米の機材を導入して進歩していった．

1985年には，コンピュータで出版を行うDTPという概念を提唱したAldus社(後にAdobe社が買収)がPageMakerを開発し，出版は新しい時代に入った．以来，PCで文書を作り，印刷，出版するのが当然になった．現在，文書作成では，マイクロソフトのWordが，そして印刷業界においてはQuark Expressが圧倒的なシェアをもっている．ほかもほとんどすべてのソフトが外国製だ．PCにしたところで，もともとIBMのATコンパチマシンの延長線上だ．つまるところ，PCとソフトウェアは，欧米文化に合わせて進化してきたのだ．

振り返ると過去のOSやアプリケーションの日本語化は，すべて妥協のうえに成り立っていたことが思い起こされる．最終的にWindowsや世界的なI18N(Internationalization)の流れで，

日本語ワープロや日本語エディタは，ほとんど消滅した．インターナショナルバージョンの登場によって，東洋の文化は壊滅的打撃を受け，暗黒の時代に突入している．

この状況は，日本人にとって苦痛以外の何物でもない．たとえば，PCの画面は絶対的に読みづらい(フォントの質が悪い，表示ドット数が不足)，画面の中で目に入る文字数が少ない，スクロールすると眼が疲れる，多くのメニューのせいでマルチウィンドウが使いづらい，などだ．画面が縦長に使えたら……．ページが実際の本に近い感覚でめくれたら……．便利なしおりの機能があったら……．気楽に線が引けたら……．マウスがもっと使いやすかったら(あるいはもっと優秀なポインティングデバイスがあったら)などと考えることは多い．また，文書を読むだけなのに，なぜ文書作成機能をもった重たいソフトを使うのかという疑問もわく．この種の問題解決策として，過去にNECや富上通などが何度か挑戦したが，いずれも成功せずに中座している．

その結果としてある程度の解決策を示しているのがAcrobat Readerだ．これを使えば，本を読むのに，より近い形で画面に表示することができる．しかし，出力文書に造形美はあっても機能美が不足している．また，スピードが遅すぎるし，何より欧米製なのが気に入らない．

● 日本発の新しい出版文化を！

特筆できるのは，日本の新聞の印刷技術かもしれない．1959年，毎日と読売の両紙が全自動テレタイプ式の印刷を開始した．以来，読みづらい日本語というハンディを回避するために，より読みやすい扁平活字体を開発し，高品質のフォントを追求，字詰めの方法や流し込みの規則などで多くの研究開発・改良がなされている．1957年より発行された「新聞印刷技術」(現新聞技術)という機関誌には，次々と新しい技術や改良が発表され，今でもその勢いは衰えてはいないそうだ．これは，現代の日本が，情報化時代になることで印刷物の必要性が増したことを証明している．かつて，東洋で活版印刷が進歩しなかった二つの理由のうち，最大の一つが間違いなく解消されているのだ．

とすれば，PCでも日本語表示のための大幅な改良があってもよいのではないか．21世紀に入り，動画を次世代の開発照準にあわせている日本メーカーも多いが，出版はどうだろうか．古いテーマだが，印刷しないで，PCで読める本というのも画期的

な気がする．パソコンの性能が上がった今こそ，再考できる気がする．ハードウェアに限界があるにせよ，それをカバーするのがソフトウェアのはずだ．それでもPCが読書に向かないとすれば，何が足りないのかを研究してもよいのではないか．

いつでもどこでも読める，何回でも読む気になる．それが現代の「読書百遍」の心だ．東洋の心をもった日本発のPC出版文化というのも素晴らしい．

PCの台数は，すでに世界で十億台を越えた．インターネット人口は，2010年に2億人を越すといわれている．しかし，その圧倒的多数が東洋圏だ．欧米が完全にイニシアチブを握り，情報発信元となっている現在の状況は，あまりにも異常だ．

これからは新しい技術が再び東洋，日本から，欧米に飛び出していくことがあってもよいはずだ．13世紀の王禎のように，東洋発の新技術を，欧米に普及させ，思い知らせてやることはできないものだろうか．

あさひ・しょうすけ　テクニカルライター
イラスト　森　祐子

Engineering Life in

シリコンバレーに夫婦で出向(第一部)

■今回のゲストのプロフィール
鈴木友子(すずき・ともこ):1992年,愛知教育大学総合科学課程国際文化コース英米文化選修卒業.NECマイコンテクノロジーに入社後,社内でのOn the Job Trainingを受け,組み込みマイコン用ソフトウェアツールのサポート業務に就く.8～64ビットマイコン向け開発ツールの関連業務を8年間行った後,2001年10月にNEC Electronics America, Inc.にProduct Marketing Engineerとして出向し,現在に至る.趣味は油絵,スキー,キーボード.最近買ったDVDは「007」7本セット,最近買ったお気に入りの本は『気がつくと机がぐちゃぐちゃになっているあなたへ』.
鈴木　敦(すずき・あつし):1988年,幾徳工業大学(現神奈川工科大学)卒.同年NECマイコンテクノロジー入社.入社以降十年以上にわたり,NEC独自アーキテクチャの32ビットCPU向け基本ソフト開発に従事.2001年に米国出向となり,業務内容も開発からサポートへと変わり現在に至る.趣味はドライブ,スキー,読書.日本ではバイクにも乗っていたが,こちらではいまだに免許も取得していない.

☆ 条件がそろい,二人で渡米する

トニー さて,今回は「初」が二つになります.出向型でシリコンバレーに来られている方の初登場と,夫婦で対談の初登場です.まずは,エンジニアリングの世界にどういう経緯で入られたかについて話していただけますか?

友子 私の場合は英米文化を勉強していたので,大学での専攻の話をするとびっくりされる場合が多いですね.アメリカの大学では,かなり即戦力になるような教育がなされているので,理科系の専攻でない場合は不思議に思われます.そして,当時日本ではバブル経済が弾けた頃だったので,理科系以外の分野からもエンジニアを採っていたと思います.それで個人的にコンピュータなどの知らない世界に興味があったので,この分野に入り少しずつ知識を付けました.余談ですが,大学の専攻が理科系でなかったので,アメリカ入国のビザ申請には苦労しました.

トニー なるほど……シリコンバレーでも純技術系以外を専攻した人が,凄いプログラマになるという例がたまにあります.ビザは,アメリカの一般的に技術職で来る人は,最低でも大卒で理科系を専門としていたことが前提となっていますよね.

敦 僕の場合は,大学にちょうど情報工学部ができたので入りました.会社に入社したときは,さまざまな背景の人を採っていました.実際,私の同期でも畑違いの哲学部出身にも関わらず,ソフト開発で適性を発揮している人がいました.大学の専攻や学科と,ソフト開発への適性はあまり相関がないという印象がありますね.

トニー アメリカなら情報・電子工学系以外に物理,化学そして他の工学部から情報・電子系のプログラマやエンジニアに転進する人が多いですね.以前私が勤めていた会社の研究開発部のシニアエンジニアには物理や化学系の人が多かったという記憶があります.たまに理科系の背景のないエンジニアに遭遇しますが,かなり特徴のあるタイプが多いですね.

敦 僕は,その人の大学での勉強よりも,その人の個性が反映して凄いと思われるのだと思います.とくにプログラミングはセンスも重要な分野だと思います.

鈴木　敦氏

トニー さて,話は戻りますがシリコンバレーに来られて1年ぐらいですよね?それでカップルで出向というのは珍しいのでは?

友子 もともと私のほうがV_Rシリーズのマイコンまわりのサードパーティーツールサポートの仕事をしていて,海外のさまざまなベンダーと取り引きがありました.V_R以前の業務でも,アメリカに何度か出張で来ることがありました.それで,まずは私のほうにシリコンバレーへの出向の話がありました.

敦 僕の場合は,V800シリーズのマイコン開発ツール,具体的にいうと言語系のツールの開発をやっていました.これをかなり長い期間やっていたので,少し違った観点からの仕事もしてみたいと思いました.ちょうど出向の話も一致して,こちらでサードパーティとやり取りをする仕事に就くことになりました.

友子 日本では,同じ会社でも職場が違いました.結婚して1年ぐらいだったので悩みや葛藤もありましたが,今回はお互いに条件と背景がそろっていたので,一緒に来ることができて良かったと思います.

トニー なるほどね.出向というと配偶者がオマケ的な存在ですが,お二人とも仕事を続けられるのは良い体験になりそうですね.

敦 シリコンバレーの会社にも出向はあるのですか?

トニー ありますが,大きな違いは,帰って来る際の保証がいっさいないところでしょうか.つまり,戻りたい場合にはシリコンバレー側の会社,つまり本社でのポジションは自力で確保するという点です.戻りたい場合には,自己アピールして自分の戻る場所を作る必要があります.また,もともと共働きのカップルが出向することもありますが,配偶者が仕事を辞めてしまうことが多いです.二人のタイミングを合わせるのが原因のようです.たとえば,配偶者のほうがさっさと会社を辞めていて,出向がキャンセルになったという例もあり,会社側は何もしてくれないので厳しいと思います.

☆ 想像していたシリコンバレーとのギャップ

トニー さて,こちらに来られていろいろと新しい発見やギャップを感じられたかと思うのですが,どうでしょう?

友子 シリコンバレーは意外とアジア系の人が多いことに驚きました.中華レストランや韓国レストランで急に中国語や韓国語で話しかけられたり,また,これらのレストランで英語が通じない場合があるので,たまに苦労します.

トニー シリコンバレーの生活に,英語以外に中国語や韓国語が必要だったり(笑).

Silicon Valley

H. Tony Chin

対談編

敦 職場は日本の会社ですから，普通のシリコンバレーの会社よりも日本人が多いとは思うのですが，僕の上司は台湾人だし同僚もインド人，中国人，アメリカ人とさまざまです．

それに加え，中国人や韓国人がこちらに来ても自分達の文化を崩さずに日々の生活を送っているところはエネルギッシュに感じます．

友子 そうそう，日本人にない傾向ですね．日本人だと日本食を食べることぐらいでしょうか？　他のアジア系の人達はどこかに絶対にミックスされない自分の文化をもって生活しているように見えます．

トニー たしかに日本と他のアジアを比較すると，日本のほうが民族意識がそれほど高くないこと，他のアジア系の人達が自国に帰らない移民が多いことに気付きますね．日本からはどちらかというと一時的な引っ越しみたいな意味が多いと思います．だからよりいっそう他のアジア系の人達は自分たちの文化の意識が強くなるのだと思います．

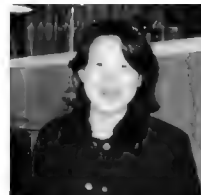
友子 あとは，シリコンバレーが意外と田舎なのにはびっくりしたのと同時にほっとしました．平屋が多いし，すぐ近くに山とか牧場があって牛がノンビリと草を食べている光景が見えますよね．そこら中の木にリスとかもたくさんいるし，自然や動物が身近に非常に多いという印象があります．

敦 僕もシリコンバレーというと「技術の最先端をいく近代都市」みたいなイメージがありましたから，ビルがほとんどなく，せいぜい2階建てぐらいのオフィスビルばかりなのにはびっくりしました．

トニー う～ん，なるほど……．ほんの数年前までは果樹園を中心とした農業地帯でしたからね．そういう意味ではシリコンバレーはサンフランシスコに対して，非常にコンプレックスをもっている都市なんですよ．人口や生産売上高ではアメリカを代表するハイテクの街なのに，北カリフォルニアの中心というとサンフランシスコなんですよね．

敦 コンプレックスというと具体的には？

トニー たとえば，プロスポーツのチームがあると全米のメディアでも知名度が上がるでしょう？　シリコンバレーだと，やっとできたサンノゼ・シャークスのホッケーチームぐらいです．やっぱり3大スポーツの野球，アメフト，バスケットのどれかがないとメジャーな都市じゃないって感じですね．もう過去20年近くサンフランシスコ・ジャイアンツをサンノゼのほうに引っ越すようオーナーに働きかけたり，野球場を作る計画を立てたりなどいろいろ動きはありますが，まだまだ実現されていない．あとは，シンフォニーとかバレエとか著名なカルチャーイベントを主催するとかね……．日本でいうと東京と川崎の関係みたいなものでしょうか？　川崎市の方には申し訳ありませんが(笑)．

友子 う～ん，それはわかりやすい例かもしれませんね．日本を代表する電子機器メーカーが川崎市と神奈川県近辺に軒を並べてますからね．でも，私はもともと街っ子だと思うのですが，サンフランシスコのあのごちゃごちゃした街の雰囲気よりシリコンバレーのほうが心地良いです．

鈴木友子氏

次回について：引き続きシリコンバレーの印象，仕事面また私生活面での発見について話を伺う．

トニー・チン　htchin@attglobal.net　WinHawk Consulting

Column ― ストックオプションによる富の分配

ここ数年，アメリカの経済ニュースでは株のインサイダー取り引きによる不正や汚職が後を絶たない．インターネットバブル中に株価が天文学的数字に引き上げられ，実体も何も残らないインターネット系の会社もシリコンバレーに多く存在した．

最近になって，支給されたストックオプションの経済効果を研究した調査が発表された(サンノゼマーキュリー誌，1月10日発表)．結果からいうと，インターネットバブル中にストックオプションを支給された人々は，仕事の内容に関わらずかなりの富を手にしていたようだ．1ドル120円程度の換算でも，平均して日本円で軽く4000万円以上(ピークの'99年ぐらいに換金した場合を想定)の計算となる．これらは上層部から受付や事務員まで幅広く支給されており，それぞれの格差はあるにしても膨大な富を分配したことになる．また同じ研究では平均的に会社の株取得率が上層部は14%，そして一般社員では19%になっていると指摘している．2000年にバブルが弾けた当時でも，換金した場合一人あたり1500万円程度の計算となる．

この報告書ではストックオプションの民主的富の分配を絶賛しているようだが，さまざまな議論を呼ぶ結果となっている．まずは，すごく機嫌の悪いのは，高価格で株を買ってしまった投資家達だ．結局彼らの富が分配されたともいえるからだ．またストックオプションの恩恵を受けるには，株式が上場する前に入社することが必須となる．これらの社員が本当に会社の価値を上げるために貢献したかなどが問われるわけだ．一般的な傾向として，上場をめざして極端な行動に出るケースが多い(ハイテク系ではないがエンロンやWorldComの不正)．また，ストックオプションと引き換えに給料を押さえ込んだ企業も少なくないので，給料の後払い的な意味ももつといえる．

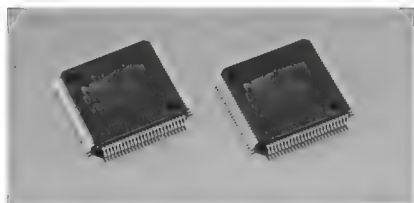
最近，ブッシュ大統領が景気対策の一つに株の配当の無課税を提案した．ほとんど配当を支給しないシリコンバレーの会社にとっては，また新しい株関連の課題となる．

HARD WARE

●32ビット1チップマイコン

V850ES/PM1

- 高分解能な16ビット分解能で6チャネル(12入力)のデルタシグマA-D変換器を内蔵.
- 電圧測定時の基準電圧を生成する基準電圧生成回路と，その電圧を出力する基準電圧出力端子を内蔵.
- システムを1チップで実現でき，システムを低コストで実現.
- 測定精度の指標である信号対ノイズ比(*S/N*)76dB(typ.)，全高周波歪み(*THD*) − 80B(typ.)のビット換算時12ビット相当の高精度を実現.
- 低消費電力で高性能なCPUコア「V850ES」を採用することで，16ビットマイコンと比較して約1/2の2.8mW/MIPSの低消費電力を実現.

■ NECエレクトロニクス(株)
サンプル価格：¥1,200
TEL：044-435-9494　FAX：044-435-9608

●16ビットCAN内蔵マイコン

MB90890シリーズ

- 二つのバンクをもつ64Kバイトのデュアルオペレーションフラッシュメモリを搭載することで，1チップでプログラム実行中に書き換え処理が可能.
- 外付け部品を不要にしたことにより，搭載面積の削減およびコストダウンを実現.
- 大容量の不揮発性データメモリを活用して，走行距離データの記録機能や，複数のシート状態を記憶しておくなどが可能.
- 動作電圧範囲を3.5～5.5Vと広く確保し，電圧低下などにも耐えられる信頼性の向上を図っている.
- 入力電圧の"L"レベル認識マージンを，電源電圧(V_{cc})の30%から50%以下に引き上げることで，雑音などに強いシステムを実現.
- CAN(Controller Area Network)コントローラを1チャネル搭載.

■ 富士通(株)
サンプル価格：¥900
TEL：042-532-1397
E-mail：edevice@fujitsu.com

●16ビットシングルチップマイコン

H8S/2628F

- CANに加え，周辺デバイスのチップセレクト付き高速同期式シリアル通信インターフェースを内蔵し，最大動作周波数を24MHzでフラッシュメモリを内蔵したF-ZTATマイコン.
- 0.35μmプロセスを採用しており，「H8S/2000」CPUコアを搭載し，最小命令実行時間41.6nsを実現.
- CANインターフェース，SCIFに加え，チップセレクト付き高速同期式SCIFを搭載.
- 8，16，32ビットの連続転送が可能であり，最大6Mbpsまでの転送速度を実現.

■ (株)日立製作所
サンプル価格：¥1,700
TEL：03-5201-5212
URL：http://www.hitachisemiconductor.com/jp/

●スマートカードリーダ用IC

73S1121F

- 8052プロセッサおよび専用のUARTをもち，ISO-7816インターフェーススロットを二つサポート.
- GPIO×4，UserIO×8，64Kバイトフラッシュメモリ，4Kバイトユーザー RAM，USB，PINパッドインターフェースを1チップ上に集積.
- RTCを内蔵し，5×6キーボードインターフェースを装備.
- 評価ボードは，インサーキットエミュレータまたはROMエミュレータの利用が可能.
- CD-ROMには，APIライブラリ，APIエクササイザーアプリケーション，サンプルアプリケーション，ユーザーガイドなどが含まれる.

■ TDK(株)
サンプル価格：¥3,000(10個時)
TEL：03-5201-7231

●画像処理LSI

AIP-7000

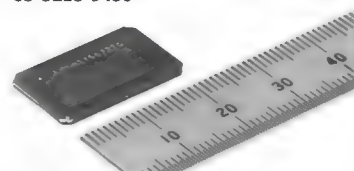
- カメラの取り込みスピード240Mバイト/sを実現する，画像処理LSI.
- ホスト転送が200Mバイト/s(32ビット/66MHz)となり，4倍の高速化を実現.
- LSI製造プロセスを微細化(プロセス0.25μm)し，パッケージサイズは304ピンのFPBGA(19×19mm)で，1枚のボード上に複数個搭載した製品も可能.
- 加減算，ビット演算(AND/OR/XOR)，乗算，積算(X/Yラインフレーム)の各画像演算をサポート.
- SDRAM(100MHz)，64/128/256/512MビットSDRAM×4のフレームメモリを搭載.

■ (株)アバールデータ
価格：下記へ問い合わせ
TEL：042-732-1030　FAX：042-732-1032
E-mail：sales@avaldata.co.jp
URL：http://www.avaldata.co.jp/

●シンクロナスSRAM

シンクロナスSRAM

- SigmaRAMに準拠し，周波数350MHzを実現した18Mビットシンクロナス SRAM.
- 36ビットチップ2個で72ビットのシステムを構築するのに比べ実装面積，動作電源電流の面で有利なうえ，動作周波数を1/2で済ませることができ，セットアップタイムやホールドタイプなどのタイミング設計に余裕を持たせることが可能.
- 最大350MHzの動作周波数の実現により，最大25Gbpsのデータ転送が可能.
- 書き込み動作にダブルレイトライト方式，読み出し動作にパイプラインリード方式を採用し，読み出しと書き込み動作を交互に行っても，データバスが衝突することがないため，デッドロックサイクルが発生せず，データバスを100%活用することが可能.

■ 三菱電機(株)
サンプル価格：¥12,000
TEL：03-3218-9450

●非接触ICカード用システムLSI

MN103S41H

- 国際標準ISO/IEC14443タイプBの非接触インターフェース仕様と，住民基本台帳カード仕様への完全準拠を実現．
- 電子署名法に適合した公開鍵暗号RSA/楕円暗号の高速処理を実現するコプロセッサを搭載し，ICカード内部で署名の生成，検証，暗復号化を高速に実行可能．
- 秘密鍵がカード外に露出しないため，高度なセキュリティの確保を実現．
- Javaアプレットがダウンロード可能で，カード発行後にサービス機能の追加が可能．
- JavaCardの機能によって，アプレット間のセキュリティが確保され，安全な機能拡張が可能．

■ 松下電器産業（株）
サンプル価格：下記へ問い合わせ
TEL ： 075-956-9998
E-mail ： semicon-press@mrg.csdd.mei.co.jp

●サラウンドサウンドCODEC

CS42516，CS42526 CS42518，CS42528

- 6および8チャネルの2種類を用意し，それぞれについてダイナミックレンジが110dBおよび114dBという2種類のパフォーマンスをもつ．
- D-Aコンバータのすべてのチャネルでディジタルボリュームコントロールと差動アナログ出力が提供され，各デバイスでは192kHzのサンプリング周波数をサポート．
- ステレオA-Dコンバータも装備されており，シングルエンドまたは差動のアナログ入力に対して個別のチャネルゲインコントロールを行うことが可能．
- さらに2系統のステレオA-Dコンバータを追加することによりマルチチャネル入力をサポートすることが可能．

■ シーラス・ロジック（株）
サンプル価格：$3.93～$6.97（10,000個時）
TEL ： 03-5226-7378　FAX ： 03-5226-7677

●A-Dコンバータ

LTC1861L LTC1865L

- 小型の10ピンMSOPパッケージで供給される3V，2チャネル，12ビットおよび16ビットのA-Dコンバータ．
- 3V電源時に150kspsでの消費電流は450μA．無変換時に自動的にシャットダウンするため，1kspsでの消費電流を3μAまで低減可能．
- 電源は，単一2.7V～3.6V．
- 2本のマルチプレクスチャネルまたは1チャネル．
- 10ピンMSOPおよび8ピンのSOパッケージで提供．

■ リニアテクノロジー（株）
サンプル価格：¥340～（1,000個時）
TEL ： 03-5226-7291　FAX ： 03-5226-0268

●16ビットトランシーバチップ

74ALVCH16245

- 3.6V I/Oトレラントを装備した低電圧16ビットCMOSトランシーバチップ．
- 最大伝播遅延は3ns（V_{CC}=3.0～3.6V），ドライブ能力は24mA（V_{CC}=3.0V）以上，スタンバイ時の電力消費量は40μA以下という特性を持つ．
- サブミクロンCMOSシリコンゲートと5層の銅線配線から構成．
- 動作範囲はV_{CC}=1.65～3.6Vで，3.6V信号のインターフェースを装備．
- 74シリーズ16245デバイスとピン互換があり，バスを効果的に切り離すイネーブル入力をもつ．
- アクティブバスホールド回路は，使用されないあるいはオープンな状態のデータ入力をロジックレベルに保つ．
- 対策回路が装備されているために，2000VまでのESDや過電圧から，すべての入出力を保護する．
- 全入出力に対するパワーダウン保護機能を備える．

■ STマイクロエレクトロニクス（株）
価格：¥120（1,000個時）
TEL ： 03-5783-8240　FAX ： 03-5783-8216

●ディジタル温度センサ

TC77/TC72

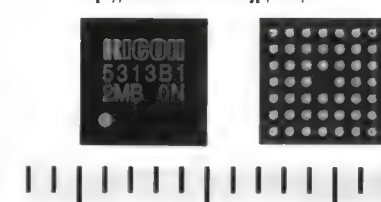
- 高精度，高分解能のディジタル温度センサ．
- 外部構成部品を必要とせずに温度を読み取ることが可能で，3ワイヤ（TC77）ならびに4ワイヤ（TC72）の標準インターフェース（SPI）により温度データを取得．
- 動作電流250μA（typ.）を実現しており，パワーシャットダウンモード（1μA，typ.）と併用すれば，電池寿命の延長が可能．
- TC72は－55℃～＋125℃までの温度を測定することができ，2.65V～5.5Vまでの作動電圧範囲が特徴．温度分解能はビットあたり0.25℃で，温度精度は－40℃～＋85℃までの温度幅に対して最大2℃となっている．
- TC77の温度分解能は，ビットあたり0.0625℃で，精度の高い温度検出と精密な温度測定が可能．－55℃～＋125℃までの温度を測定することができ，2.7V～5.5Vまでの作動電圧範囲を実現．

■ マイクロチップ・テクノロジー社
価格：下記へお問い合わせ
URL：http://www.microchip.com/

●システム電源ASSP

R5313Bシリーズ

- 低電圧化が進むCPUコア用の電源としてDC-DCコンバータを採用可能．
- CPUがスタンバイ状態の時には制御ピンによる切り替えで，DC-DCコンバータからより消費電流の少ないボルテージレギュレータへと切り替えることができ，電圧供給の状態を保持しながら電源内部の消費電流を200μAから40μAまで低減．
- DC-DCコンバータの発振周波数は900kHz（typ.）で，－20℃～70℃の範囲で865kHz～930kHzの精度を保証．
- 高周波ノイズに敏感なRF回路の電源であるLDOまでを1チップに内蔵．

■（株）リコー
サンプル価格：¥600
TEL ： 045-477-1706
E-mail ： lsi-support@ricoh.co.jp
URL：http://www.ricoh.co.jp/LSI/

HARD WARE

●クロックマネジメント

TeraClock ファミリ

- 入力用5本，出力用4本の設定可能なシングルエンド信号または差動信号を備えた，広範囲なI/O標準間での変換能力をもつ.
- 製品間での相互移行を容易にするピン互換性をもつ.
- PLL機能による信頼性の高い信号を提供.
- 冗長クロックとヒットレススイッチオーバ機能を活用し，システムのメインクロック信号が中断された場合でも，システム全体の完全性と信頼性を維持，確保.
- ネットワークルータ，ワイヤレス3G基地局，SANなど次世代通信アプリケーションで要求される高速I/Oを標準でサポート.
- サイクルごとのジッタが50ps，出力スキューが100psでありながら，最大250MHzの周波数で動作.

■ 日本IDT（株）
サンプル価格：$4.50～$16.25（10,000個時）
TEL：03-3221-6726　FAX：03-3221-5456

●周波数可変オシレータ

VariClock

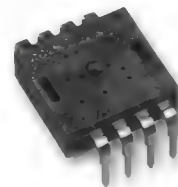
- 市販のオシレータ基板パターンにそのまま挿入可能で，最大400MHzまでのクロック信号を1MHzの分解能で発生する周波数可変タイプのオシレータ.
- モジュール上に実装した3個のロータリスイッチによって，クロック信号の周波数を変更することが可能.
- 23×31mmの大きさで，重さ15g以下と小型軽量を実現.
- 汎用14ピンDIPタイプのオシレータ用基板パターンに直接挿入可能.
- LVTTLレベル出力で，5V-TTLに対応可能.
- 10MHz～400MHzのクロック信号を発生可能で，広範囲な周波数レンジをサポート.
- 周波数誤差は，±0.1%以下の高精度.
- ボード上のジャンパソケットの切り替えにより，5V/3.3Vの両電源に対応.
- 外部イネーブル信号により，発振の強制停止が可能.
- 周波数の設定状態を随時確認できるLEDを搭載.

■（株）DesignGateway
価格：¥19,500
TEL：03-6644-1121
E-mail：info@dgway.com

●光学式マウス用ポジションセンサ

ADNS-2610 ADNS-2620

- パソコン用のマウス，トラックボールおよびその他入力装置などのエントリモデル向けのポジションセンサ.
- パッケージモールド部分のサイズを9.9×9.1×4.6mmとし，8ピンのStaggered DIPで提供することで，部品占有面積を従来品の1/2以下に低減.
- 2線式シリアルポートによって，ポジションセンサのパラメータ設定と読み出し，およびマウスの動作情報の読み出しを行う.
- 最大400cpiの分解能を有する.
- パワーダウン機能により，マウスを使わない場合の消費電力を抑制可能.

■ アジレント・テクノロジー（株）
サンプル価格：¥630（ADNS-2610）
¥700（ADNS-2620）
TEL：0120-61-1280

●小型VCSO

EV-3104（電圧制御型SAW発振器）

- 2GHz帯（2～3GHz）を直接発振させ，出力する小型VCSO.
- 高密度化した高周波数回路技術と低損失のダイヤモンドSAWにより，小型，低ジッタ，低位相ノイズを実現.
- 直接発振した周波数をそのまま出力することで，位相ノイズ特性が従来の回路方式と比較して優れており，数GHz～数十GHz帯の出力回路設計を実現.
- 次世代テレマティクス無線通信システム，無線アクセスシステム，次世代光ネットワーク機器などに適する.

■ セイコーエプソン（株）
価格：下記へ問い合わせ
TEL：042-587-5878
E-mail：ED_QD_Marketing@exc.epson.co.jp

●バリキャップダイオード

HVL355CM/HVL358CM/HVL381CM/HVL396CM/HVL397CM

- 携帯機器向けに，小型1006タイプで厚さ0.4mmを実現したTEEPパッケージを採用.
- アウターリードを含めて1.0×0.6×0.4mm（max.）を実現.
- リードがパッケージの外に設けてあるため，実装後のはんだフィレットを目視にて確認可能.
- 本パッケージを採用した携帯電話などのアンテナスイッチ用PINダイオード「HVL142AM」，「HVL144AM」を製品化.

■（株）日立製作所
価格：¥10～¥12（10,000個時）
TEL：03-5201-5241
URL：http://www.hitachisemiconductor.com/jp/

●表面実装型ポリスイッチ

SMD050-2018

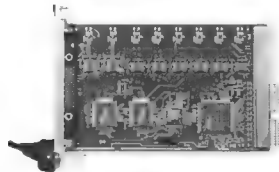
- テレコムやネットワーキングアプリケーション向けの表面実装型ポリスイッチ.
- VoIPやパワードEthernet機器向けに，リセッタブルな過電流保護を提供.
- IEEE802.3af Ethernet仕様の電圧および電流条件に準拠.
- パワードEthernet機器の電源，出力間に実装することにより，電源管理回路に2重の保護を実現.
- パワー制御用FETがショートした場合でも個々のポートを保護し，システム全体の電源トラブルを防止.
- 1回かぎりのヒューズにあるような疲労切れ，誤動作，実装時のヒューズ切れなどの問題を回避可能.

■ タイコ エレクトロニクス レイケム（株）
サンプル価格：¥30（100,000個時）
TEL：044-900-5110　FAX：044-900-5140

●高速シリアル通信 CompactPCI モジュール

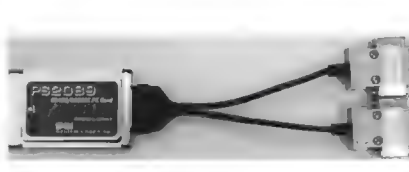
TCP866

- ドイツの TEWS TECHNOLOGIES 社が開発した，高速シリアル通信ボード．
- 通信ボーレートは 25 ～ 921.6kをカバーし，動作温度は－40 ～＋85 ℃．
- ESD プロテクショントランシーバを使用．
- RS-232-C または RS-422 を 8 チャネルサポート．
- J2 I/O オプションをサポート．
- PICMIG 2.0 Rev2.0 準拠の，3U 32 ビット標準 Compact PCI モジュールを採用．
- VxWorks，pSOS，Linux，Windows2000 など各種 OS のドライバオプションを用意．

■（株）ナセル
価格：¥131,000
TEL ：03-5921-5099　FAX ：03-5921-5098
URL：http://nacelle.co.jp/

●PC カード

PS2089

- PC カードスロットに挿入して使用することで，RS-422 または RS-232-C 通信ポートが，2 チャネル増設可能になる PC カード．
- ケーブルを変えることで，RS-422 から RS-232-C へのチャネルごとの切り替えが可能．
- Windows 付属の標準ドライバで使用可能．
- I/O のリソースは，自動的に割り付けられる．
- カード外形は，TYPE Ⅱサイズ準拠．
- 使用環境は，JEIDA Ver4.2/PCMCIA2.1準拠．
- 最大通信速度は，非同期 230.4kbps を実現．
- 使用温度は 10 ～ 35 ℃，湿度は 20 ～ 80%．
- ＋5V 300mA 以下の電源供給．
- カード部の重量は，30g 以下．

■（株）システムクラフト
価格：¥35,800
TEL ：042-527-6623　FAX ：042-527-3079
E-mail ：sdo@scinet.co.jp
URL：http://www.scinet.co.jp/

●シグナルプローブソケット

FFP1152-S2-Ⅱ

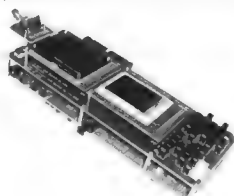
- （株）エスケーエレクトロニクスとの共同開発による，FPGA/PLD 開発支援用 FBGA IC ソケット．
- Xilinx 社製の FPGA Virtex-Ⅱ FF1152 に最適化され，回路開発でのシステムデバッグ，評価，解析において作業効率の大幅な低減を実現．
- 1mm ピッチで 1152 プローブ端子が 35mm 角内にマトリクス配列し，中間層のシグナルプローブ基板を貫通させた構成の FBGA 用ソルダーレス IC ソケット．
- プローブ基板内層配線は，ファインパターン技術を駆使した 60μm 加工幅を採用し，1 本ごとにグラウンドガードさせることで，クロストークにも配慮している．

■（株）エス・イー・アール
価格：¥450,000
TEL ：03-5796-0330　FAX ：03-5796-3210
E-mail ：ser@ser.co.jp

●開発プラットホーム

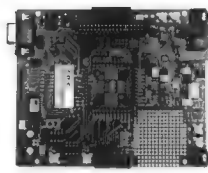
SH7300 統合プラットホーム

- 動作周波数 118.8MHz の SH7300 を搭載したアプリケーション開発用のボード．
- 携帯電話のアプリケーション開発に必要な機能やインターフェースなどをコンパクトに実装．
- VGA サイズの CMOS カメラモジュールや 2 枚の液晶パネル，30 個のキースイッチなどを 100 × 254mm の小型サイズに収納．
- オーディオインターフェースや拡張スロットを備えているため，外部デバイスの接続が可能．
- OS（μITRON）やデバッガを含むソフトウェア群を用意することで，マルチタスクアプリケーションの評価を行うことができ，評価期間の短縮を実現．

■（株）日立製作所
価格：下記へ問い合わせ
TEL ：03-5201-5234
URL：http://www.hitachisemiconductor.com/jp

●FPAA テストボード

形 2SAA120D04 FPAA テストボード

- アナログ IC の回路設計を短時間で実現する設計支援ツール．
- アナログ IC を試作できる FPAA（Field Programmable Analog Array）チップを搭載しており，パソコン画面上の設計ツールを用いて機能ブロックを選択するだけで，アナログ IC の機能を完成させることが可能．
- 同社従来品と比較して，微小信号処理能力を 100 倍，動作速度を 4 倍に向上．
- 乗算機能や任意波形信号生成機能などを追加することで，設計可能なアナログ回路の範囲が増大．
- 信号入出力部のノイズ防止フィルタを内蔵化することで，周辺部品点数の低減を実現．

■オムロン（株）
価格：オープン価格
TEL ：075-344-7074
URL：http://www.omron-ecb.com/sc/fpaa/

●ファンクションジェネレータ

FG-281

- 正弦波，三角波，方形波などの基本波形の発振，デューティ，オフセットなどの基本機能に加え，スイープ，バースト機能を有する DDS 方式採用のファンクションジェネレータ．
- 0.01Hz ～ 15MHz までの広い発振周波数範囲をカバー．
- 周波数確度は，± 50ppm．
- VFD に，電圧および周波数を同時に表示可能．
- 周波数と振幅はテンキー入力可能．
- 周波数変更時も波形が不連続にならない．
- デューティ可変範囲は，0 ～ 100%．
- 出力端解放時の最大オフセット可変量は，＋10V/－10V．

■（株）ケンウッド ティー・エム・アイ
価格：¥138,000
TEL ：045-939-7053　FAX ：045-939-6256
E-mail ：info@kenwoodtmi.co.jp

SOFT WARE

●リアルタイムOS

SMX

- 米国マイクロディジタル社が開発した，中小規模組み込みシステム向けのリアルタイムOS.
- ランタイムロイヤリティがフリーで，ソースコードが提供される.
- 高速タスクスイッチや，極小割り込み遅延を実現.
- プリエンプティブ，ラウンドロビン，タイムスライスなどのスケジューリングが可能.
- モジュール構造によりスケーラブルな構築が可能で，小さなフットプリントを実現.
- ユニークで高機能なメッセージング方式を採用.
- スタック共有モデル方式を採用.
- C++をサポート.
- ダイナミックモジュールローディングが可能.

■（株）フリーステーション
価格：下記へ問い合わせ
TEL：03-3866-2345　FAX：03-3866-2346
E-mail：sales@freestation.co.jp

●グラフィカル設計ツール

visualSTATE Target

- IARシステムズ社が開発した，グラフィカル設計ツール.
- グラフィカル設計言語を用いてターゲットとなるマイコン向けに，設計，テスト，ドキュメント自動生成，最適化コードの生成という機能を実現.
- ローエンドのマイコンでも，OSなしでコンカレンシをハンドリングし，コンパクトなコード設計を行うことが可能.
- 対応マイコンは，ARM/AVR/SH/PIC/M16C/M32C/V850/Z80ほか.

■（株）プロトンソフトボート事業部
価格：¥498,000（ARM用）
TEL：03-5337-6434　FAX：03-5337-6130

●ダイナミックツールパッケージソフト

CyberClocksソフトウェア

- プログラマブルクロックの設定に「ブラックボックス」アプローチを取り入れることによって，タイミング設計を簡素化.
- システム設計者がパソコン上のプログラミングシミュレーション環境で，リアルタイムに要求事項をカスタマイズ可能.
- 最適化されたクロッキングソリューションを自動的に検索すると同時に，ユーザーとの対話と設計時間を最小化.
- 要求事項の精緻な計算を実行し，設計者が設定したすべての性能パラメータが満たされるように計算結果を最適化.
- 組み込まれている設計規則チェックにより，有効プログラミング条件とデータシートパラメータのもとで，PLLシステムの安定性の実現を確保.
- レジスタ記述部とプログラミングビットのコンテンツを示すスプレッドシート状のインターフェースを作成することで，複雑なプロセスから憶測するという作業を行う.
- http://www.cypress.com/support/から無償ダウンロードが可能.

■日本サイプレス（株）
価格：無料
TEL：03-5371-1921　FAX：03-5371-1955

●バーコードコンポーネント

GrapeCity BarCode 2.0J

- PowerToolsシリーズのActiveReports，VS-VIEW 7.0J専用のバーコードコンポーネント.
- 主要な2次元バーコードであるQRCodeやPDF417に対応し，わずかなプロパティを設定するだけで，バーコードの出力が可能.
- CODE39，CODE39（フルアスキー），CODE49，CODE93，CODE128，JAN8，JAN13，EAN128，ITF，POSTNET，UPC/A，UPC/E，UPC/Eアドオン，NW-7（CODABAR），カスタマバーコードなど豊富なバーコード規格に対応.
- 帳票に組み込むことで，スムーズな管理システムを実現でき，コスト削減が可能.

■グレープシティ（株）
価格：¥19,800（ダウンロード販売）
TEL：022-777-8211　FAX：022-777-8233
E-mail：sales@grapecity.com

●キャリアグレードLinux

MontaVista Linux Carrier Grade Edition 3.0

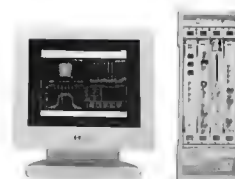
- OSDLに準拠したキャリアグレード製品であり，テレコムの可用性の要求やOSDLの機能の要求に対応するために開発され，総合的にテストされている.
- ホットスワップ，リモートブート，ディスクレスおよびコンソール操作レスのプラットホームをサポート.
- ドライバの堅牢化，ウォッチドッグタイマサポート，Ethernet冗長性と集約，RAID1サポート，ジャーナリングファイルシステムのサポート，ディスクとボリューム管理を含むハイアベイラビリティ機能を搭載.
- リソースモニタリング，カーネルクラッシュダンプと分析機能，構造化されたカーネルメッセージ，ダイナミックカーネルプロービング，ハードウェアエラーロギング，イベントへのリモートアクセスなどのサービス性を装備.
- スレッドプログラムのためのデバッガサポート，カーネルデバッガのサポート，カーネルクラッシュダンプと分析ツールなどをサポート.

■モンタビスタソフトウエアジャパン（株）
価格：下記へ問い合わせ
TEL：03-5469-8840

●無線LAN測定ソフトウェア

9600シリーズB7R 無線LAN測定ソフトウェア

- IEEE802.11bの測定機能に加えて，PBCC22（22Mbps対応）とPBCC33（33Mbps対応）の変調フォーマットに対応することで，IEEE802.11gの送信機テストを行うことが可能.
- IEEE802.11aの測定項目を自動で設定し，合否判定を行う簡易ソフトウェアを提供.
- 開発での送信項目自動設定の効率改善を図ることが可能.
- 無線LAN信号の送信レートを自動判別し，それに対応して設定を開始する機能により，より現実に近い環境で測定を実行可能.

■アジレント・テクノロジー（株）
価格：¥5,000,000～
TEL：0120-421-345

●組み込みソフトウェアコンポーネント

USNetPlus USFilesPlus

- 米国LANTRONIX社が開発したソフトウェアテクノロジ.
- TCP/IPプロトコルスタックUSNet/組み込みファイルシステムUSFilesを元に独自開発した製品で，CPUやOSに依存しない設計でコンパクトなコードサイズが特徴.
- USNetPlusは，多くのネットワークプロトコルを実装し，Ethernetを包含するなど，多彩な機器のネットワークへの対応化が可能.
- USFilesPlusは，DOS/Windows互換ファイルシステムで，リアルタイムOSを必要とせず，コンパクトな機器でデータを管理することが可能.

■（株）日新システムズ
価格：下記へ問い合わせ
TEL：075-344-7880　FAX：075-344-7901

●認証サーバソフトウェア

Enterpras Lite 1.0

- RADIUS認証サーバのエントリモデルとして，無線LANの認証に特化した認証サーバソフトウェア.
- 中小規模のネットワークで無線LANを安全に構築するために必要な機能だけを絞り込んでいる.
- 稼働環境としてはオープンプラットホームであるLinuxを採用しながら，ユーザー管理はWindowsドメインをそのまま利用可能.
- 無線LANのセキュリティ確保は，IEEE 802.11xを利用.
- EAP-TTLS（PAP）によって，クライアントごとにディジタル証明書を用意しなくても，高いセキュリティ強度を得ることができる.
- 規模に応じたリーズナブルな導入コストを実現するため，登録できるアクセスポイントの数に応じたライセンス体系を採用し，実験的なスモールスタートにも対応.

■（株）ステラクラフト
価格：¥100,000～¥900,000
TEL：06-4799-3333　FAX：06-4799-3330

●ドキュメント自動生成ツール

A HotDocument

- JBuilder，WebSphere Studio，Sun ONE Studio，Oracle9i Jdeveloperの各製品に対応したドキュメント自動生成ツール.
- 主要なJava統合開発環境に対応したことにより，Visual Studio.NET，OfficeXP対応版の同製品で自動生成されたドキュメントの統一を図ることができる.
- システム開発で使われているほとんどの開発言語から，すべて同じ形式の納品ドキュメントの生成が可能.
- ソースファイルより20種類以上のドキュメントを生成.
- Java標準ドキュメント生成ツールJavaDocでは得られないメソッド一覧表，インターフェース証明書，コメント行，実行行の情報などを出力.

■（株）ハローシステム
価格：¥39,800
TEL：03-5367-5183　FAX：03-5367-5181

●電磁場解析ソフトウェア

PakSi-E/PakSi-TM

- 米国オプティマル社が開発した，集積回路パッケージの伝送線路で問題となる信号劣化とそれにともなう不要ノイズ解析を行う電磁場解析ソフトウェア（PakSi-E）およびパッケージの熱/疲労解析ソフトウェア（PakSi-TM）.
- PakSi-Eは，リード，ワイヤボンディングを含めた抵抗，インダクタンス，容量などの寄生パラメータを抽出．設計上問題となるクリティカルネット，あるいはパッケージ全体を解析．パッケージ内のレイアウト設計の前段階でデザインルール策定のための解析を実行．寄生パラメータから特性インピーダンス，順方向/逆方向クロストーク比，遅延，減衰，Sパラメータ，反射係数，定在波比，奇/偶/差分伝送モードのインピーダンス定数を算出.
- PakSi-TMは，自然対流，強制対流条件での熱抵抗を計算．ヒートシンクを装着した場合の解析も可能．吸湿によるリフロー時のパッケージクラック発生の可能性を材料破壊靱性の観点から検証可能.

■サイバネットシステム（株）
価格：¥4,200,000～
TEL：03-5978-5460　FAX：03-5978-6081
E-mail：optimal-info@cybernet.co.jp

●テレフォニーネットワークアナライザ

J6844Aテレフォニー・ネットワーク・アナライザ

- IP電話の開設時およびサービス提供時に不可欠な音声品質と接続品質の評価を一台の測定器で行えるVoIPアプリケーション用解析ソフトウェア.
- 「J6800A」および「J6801A」などのネットワークアナライザ用のソフトウェア.
- IPパケットレベルでリアルタイムMOS値およびR値の算出，機器特性に合わせたMOS値およびR値を算出することが可能.
- LANだけでなくWANのインターフェースにも対応しており，ネットワークの大規模化にも対応.
- ネットワークの運用中に音声解析，シグナリングのエキスパート解析ができるほか，SIP，H.323のみ対応したシグナリングのエキスパート機能を搭載.

■アジレント・テクノロジー（株）
価格：¥1,240,000
TEL：0120-421-345

●Ethernetテストソフトウェア

TDSET2

- TDS7000シリーズディジタルフォスファオシロスコープやCSA7000シリーズコミュニケーションシグナルアナライザとの組み合わせで，IEEE802.3準拠のコンプライアンステストに対応.
- 1000Base-T，100Base-TXおよび10Base-TなどのUTPを用いるEthernetインターフェースのコンプライアンステストに1台で対応可能.
- 1ボタンでテンプレートテストやジッタテストなどが自動実行できるほか，テストレポートも自動で作成可能.
- ディジタルフォスファオシロスコープがベースであるため，回路上のEMI，ジッタ，クロストークなど設計時の問題解決にも威力を発揮.

■日本テクトロニクス（株）
価格：¥508,000
TEL：03-3448-3010　FAX：0120-046-011
URL：http://www.tektronix.co.jp/

海外・国内イベント/セミナー情報

INFORMATION

海外イベント

2/27-28 **Advanced Microelectronic Manufacturing 2003**
Santa Clara Convention Center and Westin Hotel, Santa Clara, CA, USA
SPIE
http://spie.org/conferences/programs/03/mm/

3/3-6 **SAE 2003 World Congress**
Cobo Center, Detroit, MI, USA
SAE International
http://www.sae.org/congress/

3/3-7 **Design Automation & Test in Europe**
Messe Munich, Munich, Germany
European Design and Automation Association
http://www.date-conference.com/

3/10-14 **HDI EXPO**
San Jose Convention Center, San Jose, CA, USA
PCB Design Conferences/HDI Expo
http://www.hdiexpo.com/

3/12-14 **SEMICON China 2003**
Shanghai New International Expo Center(SNIEC) Shanghai, China
SEMI
http://events.semi.org/semiconchina/V33/index.cvn

3/12-19 **CeBIT 2003**
Hannover Exhibition Center, Hannover, Germany
Deutsche Messe AG
http://www.cebit.de/

3/30-4/3 **The 2003 IEEE International Magnetics Conference**
Boston Marriot Copley Place, Boston, MA, USA
IEEE
http://www.intermagconference.com/

国内イベント

2/26-28 **IP.net JAPAN 2003**
東京国際展示場(東京ビッグサイト，東京都江東区)
リックテレコム
http://www.ric.co.jp/expo/ip2003/index.html

2/26-28 **国際ナノテクノロジー総合展・技術会議 nano tech 2003**
日本コンベンションセンター(幕張メッセ，千葉県千葉市)
nano tech 実行委員会
http://www.ics-inc.co.jp/nanotech/

3/4-7 **IC CARD WORLD 2003**
東京国際展示場(東京ビッグサイト，東京都江東区)
日本経済新聞社
http://www.shopbiz.jp/2002/t_index.phtml?PID=0003&TCD=IC

3/14-16 **Net. Liferium 2003**
パシフィコ横浜(神奈川県横浜市)
Net.Liferium 実行委員会，Key3Media
http://www.key3media.co.jp/Net-Life/

4/3-6 **ROBODEX2003**
パシフィコ横浜(神奈川県横浜市)
ROBODEX 実行委員会
http://www.robodex.org/

4/9-11 **SEMI FPD Expo 2003**
東京国際展示場(東京ビッグサイト，東京都江東区)
SEMI
http://events.semi.org/semifpdexpo/V33/

4/16-18 **TECHNO-FRONTIER 2003**
日本コンベンションセンター(幕張メッセ，千葉県千葉市)
JAPAN MANAGEMENT ASSOCIATION
http://www.jma.or.jp/tf/

開催日，イベント名，開催地，問い合わせ先の順

セミナー情報

組込みネットワーク機器を安く，早く作る方法
開催日時：2月28日(金)
開催場所：アクセレックセミナールーム(神奈川県横浜市)
受講料：無料
問い合わせ先：(株)アクセレック，☎(045)475-4118，FAX(045)477-2138
http://www.axelec.net/, http://www.netsilicon.co.jp/

TCP/IP による I/O 制御の実際～Ethernet を利用した組み込み機器の設計
開催日時：2月28日(金)～3月1日(土)
開催場所：CQ 出版セミナールーム
受講料：25,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125

高速回路基板設計対応 PowerPCB V5.0.1 体験セミナー
開催日時：3月4日(火)
開催場所：SORA 新大阪 21(大阪府淀川区)
受講料：下記へ問い合わせ
問い合わせ先：パッズ・ジャパン(株)，☎(03)5304-5753，FAX(03)5304-5754 http://www.padsjapan.co.jp/htm/topics.html

USB On-The-Go の技術概要と市場展望
開催日時：3月4日(火)
開催場所：オームビル(東京都千代田区)
受講料：49,700円
問い合わせ先：(株)トリケップス，☎(03)3294-2547，FAX(03)3293-5831
http://www.catnet.ne.jp/triceps/sem/030304n.htm

電子機器の熱設計・熱対策の基礎
開催日時：3月6日(木)
開催場所：CQ 出版セミナールーム
受講料：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125

JPEG2000 のアルゴリズムとアプリケーション開発例
開催日時：3月6日(木)
開催場所：オームビル(東京都千代田区)
受講料：52,700円
問い合わせ先：(株)トリケップス，☎(03)3294-2547，FAX(03)3293-5831
http://www.catnet.ne.jp/triceps/sem/030306n.htm

NTSC 基礎講座(1日コース)
開催日時：3月6日(木)，3月7日(金)
開催場所：日本テクトロニクス(東京都品川区)
受講料：20,000円
問い合わせ先：日本テクトロニクススクール窓口，☎(03)3448-3015
http://www.tektronix.co.jp/News/School/main.html

無線 LAN のセキュリティ技術と対策
開催日時：3月7日(金)
開催場所：SRC セミナールーム(東京都高田馬場)
受講料：48,000円
問い合わせ先：(株)ソフト・リサーチ・センター，☎(03)5272-6071
http://www.src-j.com/seminar_no/23/23_074.htm

ISS 製品トレーニングコース RealSecure 7.x
開催日時：3月12日(水)～3月14日(金)
開催場所：日本システムハウス(東京都新宿区)
受講料：240,000円
問い合わせ先：日本システムハウス，sales2@nsh.co.jp，☎(03)3366-3101 http://www.nsh.co.jp/security/support/trainning.html

USB2.0 仕様 On-The-Go セミナー
開催日時：3月13日(木)
開催場所：みなとみらい 2-3-3 クイーンズタワー B 7F クイーンズフォーラム会議室(横浜市西区)
受講料：無料
問い合わせ先：(株)グレープシステム基本ソフトウェア事業部セミナー係，☎(045)222-3761，FAX(045)222-3759
http://www.grape.co.jp/seminar.html

USB の基礎と応用
開催日時：3月13日(木)
開催場所：CQ 出版セミナールーム
受講料：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125

PIC 入門と I/O 制御技術
開催日時：3月15日(土)
開催場所：CQ 出版セミナールーム
受講料：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125

やり直しのための積分変換
開催日時：3月22日(土)
開催場所：CQ 出版セミナールーム
受講料：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125

工程管理技法とスキル
開催日時：3月27日(木)
開催場所：SRC セミナールーム(東京都高田馬場)
受講料：48,000円
問い合わせ先：(株)ソフト・リサーチ・センター，☎(03)5272-6071
http://www.src-j.com/teiki_no/Src/pj_4.htm

日程はすべて予定です．問い合わせ先にご確認のうえ，お出かけください．

IPパケットの隙間から

闇からの呼び声

祐安重夫

配線のチェックとか，電源系統の確認とか，いろいろな理由をつけてスタッフが徐々にいなくなり，サーバ室に一人取り残されてから，そろそろ1時間が経過する．とりあえずやるべき仕事もないので，すでに締め切りを過ぎた原稿をメールで編集部に送ろうと，こうして書きはじめてみたのだが，何を書いたらいいのか．

インターネットのドメイン名というのはうまくできていて，世界中の構造も性格も，運用方法まで違う複数のネットワークを，巧妙にフラットな名前空間の中においてしまう．もっとも，最初はアメリカのARPAnetからスタートしたものなので，現在の2文字の国別のトップドメインとは別に，3文字のトップドメインが広く使用され，当初はアメリカによって独占されているように見えた．

現在ではGOV（アメリカ政府機関），MIL（アメリカ軍），EDU（アメリカの4年制大学）をのぞいて，COM，ORG，NETは取得が自由だし，インターネットの市場性が明らかになってからは，いくつかのトップドメインが追加されるようになったのは，周知のとおりである．

ところで，INTというトップドメインがあることは，どれくらい知られているだろう．Internationalの略で，国際的な組織がこれを取得でき，ITU（国際電気通信連合）が管理している．このような組織には国際連合（UN.INT）やNATO（NATO.INT）が存在する．さらにNATOは，NATOというトップドメインを別にもっている．

こうして見るとネットワークの世界も，アメリカ主導で動いていることは確かなようだ．また，国別の2文字のドメインも，ISO3166で規定された略号なのだが，イギリスは本来はGBであるはずのものをUKにしているし，複数のイギリス領が，独自のドメイン名をもっている．つまり，現在のアラブ問題を引き起こした大本である二つの国が，ここでも勝手なことをしているわけであり，テロが起きてもまったく不思議ではないという気分になる．

ところで，世の中には秘密結社（Secret Society）というものが存在する．彼らはいったい，どのようなドメインを取得するのだろうか．

実際にはフリーメーソンのように，存在が周知のものとなっている秘密結社も存在し，まるで冗談のようだが，

```
FREEMASON.COM
FREEMASON.NET
FREEMASON.ORG
```

の三つの存在が確認できた．そのほかにも企業，宗教団体，財団法人などを装っている秘密結社は，それぞれの表の顔にあわせて，適当なドメインを取得すれば済むことだ．

はたして秘密結社に広報活動が必要かどうかは判断に苦しむところだが，彼らとてそれなりの外部との連絡や交流を必要とするのは確かだろう．

この疑問に答をあたえてくれたのは，1999年4月号の本誌でそのWebページを掲載したウィルマース・ファウンデーション（Wilmarth Foundation）だった．ここのドメインはWILMARTH.SECといい，トップドメインSECには通常のDNSではアクセスできない．彼らとの交流のおかげで，アクセス方法を知ることができたのだった．

しくみはBINDに含まれるnslookupにあった．現在ではnslookupはobsoleteとされ，hostとdigとdnsqueryの三つのコマンドに移行することが推奨されている．ところでnslookupを使用していて，きわめてまれにBINDの結果と一致しないことがあることを経験した人は，それなりにいると思う．じつは，この中にそのしくみが隠されていたのだ．残念ながら具体的な内容については，ここでは述べることができない．

このしくみを知ってみると，ILLUMINATI.SEC（啓明結社）とか，DAGON.SEC（ダゴン秘密教団，The Esoteric Order of Dagon），あるいはSTARRY-WISDOM.SEC（星の智慧派）など，世の中にはとんでもない存在がいくつもあることを思い知らされた．中にはどこにどうやって，どのような回線を接続しているのかさえ不明のものさえある．このおかげで，人にはいえない仕事の領域が広がったが，その分だけ危険な目にあうことも増加したようだ．

すでにこの部屋で1人になって，2時間が経過しようとしている．窓のないこの部屋で，すでに何時間を過ごしただろう．ときどき，人間離れした足音が，ドアの向こうから聞こえてくる．この原稿を送信すれば，すでに締切を大幅に過ぎているので，そのまま印刷に回されるだろう．そうでないとしても，編集部と連絡する手段がこれ以降あるかどうかわからない．自分の運命さえ，自分では決められないという予感がある．

そもそもこの仕事を引き受けたこと自体が，間違いだったのかもしれない．これもまた，新たなSECドメインの立ち上げなのだから．YOG-SOTHOTH.SECという，とんでもなく邪悪な名前のドメインの．

私はメールの送信キーに手をのばし，人間のものとは思えない足音が，ドアの前で止まった……．

すけやす・しげお　インターメディア・アクセス

注：本稿の内容に関しましては，本号の月号表示をよく御確認のうえ，お読みいただけると幸いに存じます（編集部）．

読者の広場

2003年2月号特集「ワイヤレスネットワーク技術入門」に関して

▷実際に使っているケースや要望も含めて紹介してほしかった．単に規格のみでなく，どう使われていくのか，国によってどう使い方が違っているのかを含めて解説してほしかった．（てんりん）

［編］今回の特集は概要や規格標準化の話が中心で，その先の「実際の製品はどうなのか」までは言及できないものもあった，と反省しています．無線LANは動きの激しい技術分野でもあり，小誌ならではの「技術を深く追求する」記事企画を実現していきたいと考えています．

▷「UWB」というキーワードは何度か目にしたことがあったが，第5章のUWBの解説を読んだら，どんな状況か自分なりにつかめた気がした．（セイ）

［編］UWBは，いまかなりホットな技術の一つです．2003年1月に米国で行われたコンシューマエレクトロニクスの展示会「CES」(http://www.cesweb.org/)でも，話題の一つはUWBのチップセットだったようです．

▷会社で無線LANを利用して事務所と実験室をつなぐことを計画し，機器選定をしていたところだったので，今回の特集はユーザーとして参考になった．IEEE802.11g対応を謳う機器は発売されてきてはいるものの，WGの進行状況など，一般の人はそう調べないであろう（今回は私もそう）情報まで書かれており，動向を把握できてよかった．ところで，今回は「無線LAN」より「無線ブリッジ」を実現できればと思っているので，そのあたりの規格について少し調べようと思う．（は）

▷Bluetoothは，少し前には超有望な技術として注目を集めたものの，少し「足踏み状態」になっている気がする．特集で紹介された「At-BT」は，いまはやりのLinux上で動作するBluetoothプロトコルスタックということで，気になる製品でした．実装について，より詳細な解説を読みたかった．（ますと）

▷ミリ波帯を使った無線LANについて，特集第4章を読んで，モジュールが開発されたり実験が行われたりのレベルまで来ていることを知った．（ロール）

▷現在，ちまたで最も話題性のあるワイヤレスネットワーク技術とIP電話システムの概要が取り上げられていておもしろかった．これから希望するテーマとして，JavaやC++などを使ったエージェントシステムの構築があります．ソースリストなどもダウンロードできるようにしてほしい．（万年失業家）

▷特集はとても興味のあるところだったので，たいへん参考になりました．特にBluetooth，OFDM無線Modemに関しては，興味深く読みました．Bluetoothや無線LANに関する特集をこれからもお願いします．（福沢陽介）

▷ミリ波帯を使った伝送を身近な民生機器に適用するという記事には感心しました．こんな周波数まで手軽に利用できるようになったんですね．それにしても人が通るだけで遮断されるとは．野外に設置して，大型の鳥による妨害発生→有害鳥獣駆除などという連想が杞憂になるよう，きちんと完成させてから広めてほしいものです．（はくりゅう）

その他

▷OggVorbisについてはPC上などでの使用に関してはソフトウェアがいろいろ揃ってきているので比較的簡単に利用できます．ただ，本文中にもあった対応するコンパクトなハードウェアプレーヤがないため，今ひとつ利用が進まないような気がします．日本のメーカーで対応を予定しているところもあるのでしょうか．（玉出のタマ）

特集担当デスクから

☆お待たせしました！ 読者から問い合わせの非常に多かったUSB2.0対応ターゲット設計事例，そして組み込み機器にUSBホストを実装するという今回の特集は，いかがだったでしょうか．

☆USB2.0は480Mbpsだから，最高40Mバイト/秒は出るだろう……と考える人もいるかもしれませんが，そう簡単にはいかないようです．FX2のGPIFのようなしくみを活用しないと，高速転送は活かせません．

☆この号の編集作業中にSONYから新型クリエが発表されました．USB On-The-Goが採用され，しかも搭載されているデバイスは今回の特集でも紹介したフィリップスのISP1362だそうです．そろそろOn-The-Goも本格的に普及してくるのでしょうか．

☆WindowsやLinuxにおけるUSBドライバの作成事例はこれまでにも何度か解説していますが，いずれもOSの用意した階層化されたドライバ構造の中の一部，クライアントドライバの作成事例だけでした．しかし，組み込み機器にUSBのホスト機能を実装するとなると，それだけでは済みません．とはいえ，すべてを一から作成するのは困難です．

☆ここへきて，各社から組み込み機器向けのUSBホストプロトコルスタックがリリースされています．今後ますます非PCにUSBホスト機能が実装されていくことでしょう．

読者の広場

アンケートの結果

興味のあった記事（2003年2月号で実施）

①第1章 ワイヤレスネットワーク技術の現況
②第2章 Bluetoothプロトコルスタックの開発と検証
③第4章 60GHz帯を使った高速無線伝送技術
④第5章 超広帯域（UWB）ワイヤレス通信の基礎と動向
⑤第3章 OFDM無線モデムの基礎技術と設計事例
⑥フリーソフトウェア徹底活用講座（第6回）
⑦IP電話システムの概要と現状
⑧音楽配信技術の最新動向（第1回）
⑧InterGiga No.29
⑩Embedded Technology 2002
⑪Hyper ITRONとμITRON4.0/PX仕様の解説
⑪シニアエンジニアの技術草子（弐拾四之段）
⑬フジワラヒロタツの現場検証（第67回）
⑬開発技術者のためのアセンブラ入門（第15回）
⑮IPパケットの隙間から（第52回）
⑯やり直しのための信号数学（第14回）
⑯Universal Plug and Playアジアサミット TOKYO
⑱ディジタルオーディオボードの設計/製作
⑱開発環境探訪（第15回）
⑱Show & News Digest

特集「ワイヤレスネットワーク技術入門」についてのアンケートの結果

Q1　特集の内容はいかがでしたか？また，その理由は？

①満足（22%）
②まあまあ満足（33%）
③普通（45%）
④少し不満（0%）
⑤かなり不満（0%）

Q2　無線通信技術に興味をおもちですか？（興味をおもちの方に）それはどんな分野ですか？（複数回答可）

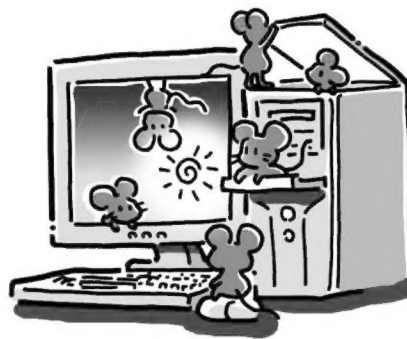
①ある（100%）
②ない（0%）
A. 高周波回路技術（11%）
B. 変復調回路技術（11%）
C. 送受信/アンテナ技術（22%）
D. 誤り訂正/圧縮など符号化（0%）
E. セキュリティ/暗号化（67%）
F. LAN/広域での運用技術（56%）
G. スペクトル拡散技術（22%）
H. CDMA/IMT-2000などの電話技術（33%）
I. 衛星通信/GPS（33%）
J. 標準規格（22%）

Q3　無線LANの規格でもっとも興味をもっているのは？

①IEEE802.11（0%）
②IEEE802.11b（22%）
③IEEE802.11a（0%）
④IEEE802.11g（34%）
⑤Bluetooth（22%）
⑥60GHz帯を使った無線LAN（11%）
⑦UWB（11%）

Interface 次号予告

2003年5月号は 3月25日発売です

生産性/品質向上のための 組み込みシステム開発技法

組み込みソフトの開発/オブジェクト指向/UML/ユースケース/エレベータモデル/電波時計モデル/コンサルテーション

いま組み込みシステム開発の現場では何が問題で，それに対しどんな解決策があるかを解説する．まず，現場エンジニアの立場から，例を示しながら，うまくいくオブジェクト指向の導入法を解説する．エレベータモデルを例に，ユースケースの記述だけで，要求される仕様についてもらさず書けることを示す．また，エレベータの機数や階数が増えたときの対処法を分析から実装まで解説する．いままでオブジェクト指向の導入に関して「定説」とされていた事柄への反証を試みる．

ソフトウェア開発を行う組織を指導・支援するソフトウェア開発コンサルタントという仕事がある．いままではオブジェクト指向やUMLを実際の開発に適用するための指導・支援を行ってきたが，最近では，開発全体の課題やビジネスゴール，事業環境，社会環境といった全体の最適化を支援するようになってきている．このソフトウェア開発コンサルタントのうち，組み込みソフトウェア開発を対象とするグループによる，組み込みソフトウェア開発の生産性/品質を向上するための方法論を特集後半で解説する．

編集後記

■持ち歩いていたノートPCを，駅周辺を歩いていて人とぶつかり，鞄ごと落とした．鈍い音がしたけれど大丈夫さ，とそのまま電車に乗り込んだ．そっとパソコンを起動したら，バックライトがつかず画面が見えない．やれやれ．修理に出して1週間．LCD交換の見積もりが来て，5万円弱の出費．痛い．しばらく経費節減生活．反省．（洋）

■知人の結婚披露パーティで司会者をつとめることになった．なぜ私が司会者？とも思ったのだが，どうやら司会者にでもしておかないと必ず何かをしでかすと思われていたからなのかもしれない．実際，板割りの演武をして「労りの心を大切に」と，その木片を配って「気配りの心を大切に」というネタを仕込んでいたのだが……(^^; (=IO)

■ここ1年ほど使ってない25型TVがあるので，知人にあげることになってまして．で，他の用事で車を借りられたので「この週末もっていけるよ」と連絡したものの，ふと動作確認のため電源を入れたら……走査線が1本しか映ってませんヨ（涙）そうだよな……もう15年くらい経つからなぁ……期待させてわるかったです>知人（M）

■HDD+DVDレコーダの複合機を買いました．機能/性能ともに満足しているのですが，本来ならば光ファイバでVODできれば個人がストレージメディアを所有する必要はなくなるはず．でも，現状では難しいようですね．一刻も早く，そんな時代がやってくることを痛切に望みます．しばらくはDVD-R地獄か…….（み）

■メモリの価格は，1～2か月の間に2倍近くも上がったり下がったりしている．個人なら買いたいときに安ければうれしい程度ですむが，大量に使用するメーカーやユーザーはどうしているのだろうか．ギャンブルと同じで，常に安く買い続けることは難しいはずなので，大きく損をすることもあると思う．利益を出すのは技術力などとは関係ないところにあるという気がする．（Y）

■捻挫しました．腫れていたけど「たかが捻挫」と思い病院にも行かずにいたら，傷みがとれないので1か月後に病院に行きました．私の話を聞いた医師は，捻挫を甘く見ない事，足を挫いた日に病院へ行く事だと言われました．診断の結果，最低1か月間は包帯などで固定するようにとの事．ああ，こんな事なら早く行っておけば良かった．（Y2）

■年末年始のTV番組．どのチャネルを見ても同じ顔と内容．特に！ラーメンを特集する番組が多いこと．ランキングだの新店だの見ているだけでお腹いっぱいになりそう．……と思うんだけど見終わった後必ず食べたくなる．最近のお気に入りは塩ととんこつ醤油．一度行ってみたいけど並んでまではねとも思ってしまう今日この頃．（ま）

■節分を過ぎ暦の上ではもう春ということですが，まだまだ寒さは厳しいようです．先日一日の消費電力が，冬場での過去最高を記録したとニュースで聞きました．そういえば暖房も電気を使うものが昔より主流だしな．でも電気代って高し，こんなに電気に頼るモノが多くて世の中いいのだろうか．（A）

お知らせ

▶読者の広場

本誌に関するご意見・ご希望などを，綴じ込みのハガキでお寄せください．読者の広場への掲載分には粗品を進呈いたします．なお，掲載に際しては表現の一部を変更させていただくことがありますので，あらかじめご了承ください．

▶投稿歓迎

本誌に投稿をご希望の方は，連絡先（自宅/勤務先）を明記のうえ，テーマ，内容の概要をレポート用紙1～2枚にまとめて「Interface投稿係」までご送付ください．メールでお送りいただいても結構です（送り先はsupportinter@cqpub.co.jpまで）．追って採否をお知らせいたします．なお，採用分には小社規定の原稿料をお支払いいたします．

▶本誌掲載記事についてのご注意

本誌掲載記事には著作権があり，示されている技術には工業所有権が確立されている場合があります．したがって，個人で利用される場合以外は，所有者の許諾が必要です．また，掲載された回路，技術，プログラムなどを利用して生じたトラブルについては，小社ならびに著作権者は責任を負いかねますので，ご了承ください．

本誌掲載記事をCQ出版（株）の承諾なしに，書籍，雑誌，Webといった媒体の形態を問わず，転載，複写することを禁じます．

▶コピーサービスのご案内

本誌バックナンバーの掲載記事については，在庫（原則として24か月分）のないものに限りコピーサービスを行っています．コピー体裁は雑誌見開きの，複写機による白黒コピーです．なお，コピーの発送には多少時間がかかる場合があります．

- コピー料金（税込み）
  1ページにつき100円
- 発送手数料（判型に関わらず）
  1～10ページ：100円，11～30ページ：200円，31～50ページ：300円，51～100ページ：400円，101ページ以上：600円
- 送付金額の算出方法
  総ページ数×100円＋発送手数料
- 入金方法
  現金書留か郵便小為替による郵送
- 明記事項
  雑誌名，年月号，記事タイトル，開始ページ，総ページ数
- 宛て先
  〒170-8461 東京都豊島区巣鴨1-14-2
  CQ出版株式会社 コピーサービス係
  （TEL：03-5395-4211，FAX：03-5395-1642）

▶お問い合わせ先のご案内

- 在庫，バックナンバー，年間購読送付先変更に関して
  販売部：03-5395-2141
- 広告に関して
  広告部：03-5395-2133
- 雑誌本文に関して
  編集部：03-5395-2122

記事内容に関するご質問は，返信用封筒を同封して編集部宛てに郵送してくださるようお願いいたします．筆者に回送してお答えいたします．

Interface

©CQ出版（株） 2003 振替 00100-7-10665
2003年4月号 第29巻 第4号（通巻第310号）
2003年4月1日発行（毎月1日発行）
定価は裏表紙に表示してあります

発行人／蒲生良治
編集人／相原 洋
編集／大野典宏 村上真紀 山口光樹 小林由美子
デザイン・DTP／クニメディア株式会社
表紙デザイン／株式会社プランニング・ロケッツ
本文イラスト／森 祐子 唐沢睦子
広告／澤辺 彰 中元正夫 渡部真美

発行所／CQ出版株式会社 〒170-8461 東京都豊島区巣鴨1-14-2
電話／編集部（03）5395-2122 URL http://www.cqpub.co.jp/interface/
広告部（03）5395-2133 インターフェース編集部へのメール
販売部（03）5395-2141 supportinter@cqpub.co.jp
CQ Publishing Co.,Ltd.／1-14-2 Sugamo, Toshima-ku, Tokyo 170-8461, Japan
印刷／クニメディア株式会社 美和印刷株式会社
製本／星野製本株式会社

日本ABC協会加盟誌
（新聞雑誌部数公査機構） ISSN0387-9569

Printed in Japan